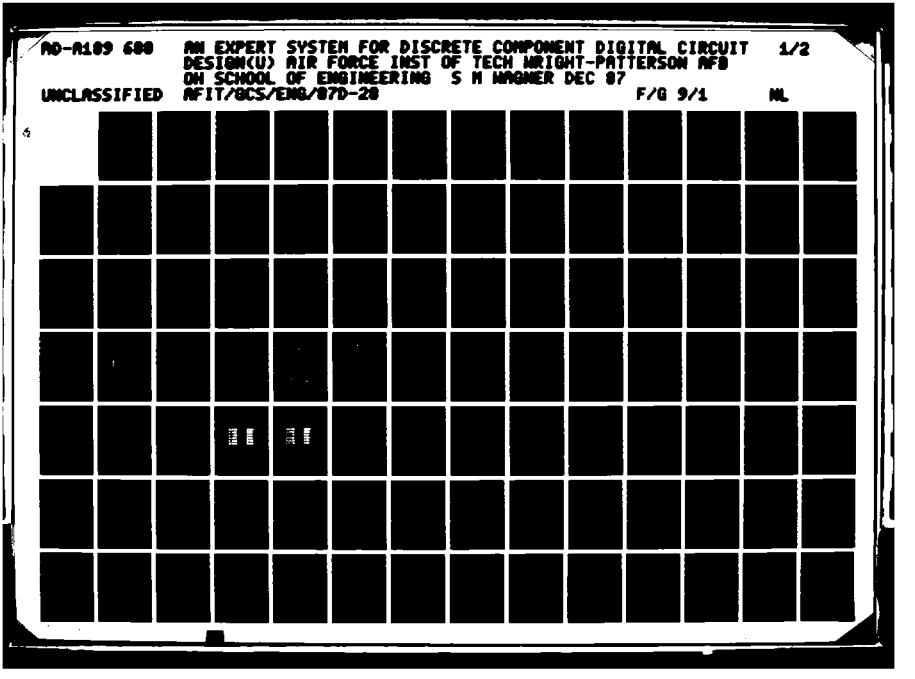
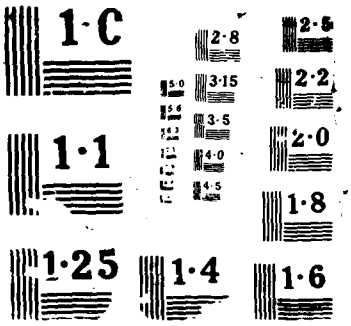
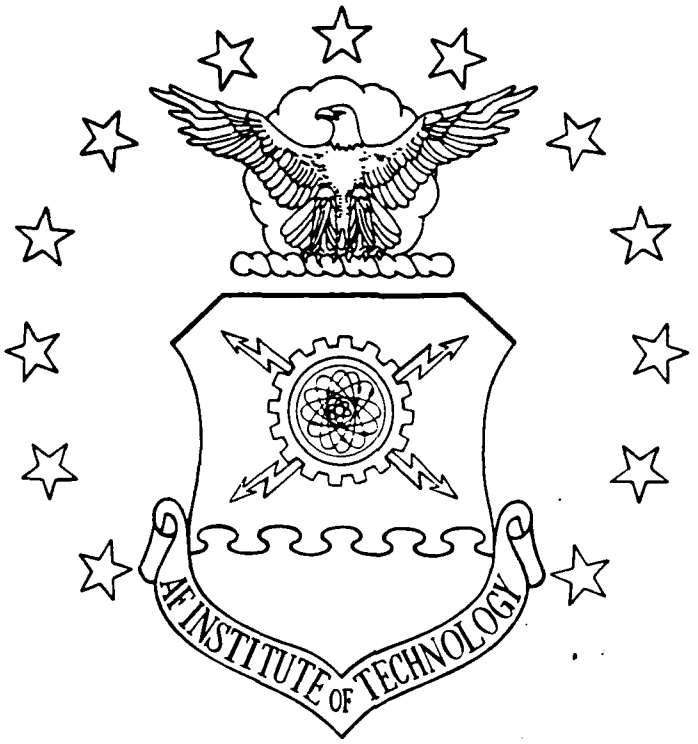
AD-A189 680 AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING S M WAGNER DEC 87 1/2

UNCLASSIFIED AFIT/GCS/ENG/87D-28 F/G 9/1 NL




1·C
2·8
2·5
3·15
2·2
3·5
2·0
4·0
1·1
4·5
1·8
1·25
1·4
1·6

(1)



AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN

THESIS

Steven M. Wagner
First Lieutenant, USAF

AFIT/GCS/ENG/87D-28

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/87D-28

AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN

THESIS

Steven M. Wagner
First Lieutenant, USAF

AFIT/GCS/ENG/87D-28

Approved for public release; distribution unlimited

AFIT/GCS/ENG/87D-28

AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Steven M. Wagner, B.S.
First Lieutenant, USAF

December 1987

Acknowledgments

The research presented in this thesis required a great deal of time and effort. Fortunately, I was assisted by a handful of people who deserve recognition. I wish to express my sincerest gratitude to my advisor, Captain Wade Shaw, for his prescient guidance and endless patience throughout my research. I also wish to thank my committee members, Captain Bruce George and Captain Nathaniel Davis, IV, for their time and cooperation in this investigation.

Special thanks are in order for Major Stephen Cross who provided me with the hardware and software necessary to implement my thesis. Also, Second Lieutenant Richard Raines for his assistance in preparing this document.

Finally, none of this would have been possible without the help of my loving wife, Cindy. Her encouragement and understanding kept me going for the last eighteen months.

Steven M. Wagner

Table of Contents

List of Figures

Abstract

The design of digital circuits is cumbersome and prone to errors. Current estimates show that a circuit designed with 5 integrated circuits has less than a 50% chance of working, due to wiring errors, when power is first applied. As the complexity increases, the probability of the circuit working on the first try decreases rapidly.

Design errors fall into two categories: wiring or logic errors. Wiring errors consist of improperly connected gates, missing connections, and violations of fanout, or race conditions. Logic errors, which are more complex to detect, require knowledge of intended circuit function.

This thesis designs, develops, and tests an expert system which decomposes digital circuits into subproblems described by database technology in order to detect wiring errors. Information needed to connect chips together is viewed as knowledge base information for the expert system. Information such as number of pins, value of each pin (input, output), type of chip, etc., are represented in a database. The interconnections between integrated circuits are evaluated within the expert system. This thesis merges an expert system tool with database techniques in a two-step approach that provides wiring assistance to a design engineer.

It is concluded that a two step process, combining the speed of database systems and the power of expert systems, results in a synergistic approach to debugging digital circuits. It is demonstrated how basic engineering techniques can be used to decompose a complex problem into smaller, solvable pieces. Performance issues such as execution speed and workload distribution are investigated. The system is integrated into an engineering workstation currently used in the Department of Computer Science and Electrical Engineering at the Air Force Institute of Technology. Suggestions for future research are considered.

AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN

1. Introduction

1.1 Background

Digital circuits can be designed using combinational or sequential logic. A combinational circuit consists of logic gates whose outputs are determined directly from the present state of the inputs, without consideration of the previous inputs. On the other hand, sequential circuits employ 'memory elements' in addition to logic gates. Their output is determined by the present inputs and the state of the memory elements. Consequently, sequential circuits depend not only on present inputs, but also the past inputs [13]. The design process of a digital circuit is very cumbersome and prone to errors. The design must be decomposed into the input and output signals, boolean functions, and logic diagrams before it is ready for implementation. Mano [13] describes the "classical method" for implementing digital circuits as having the following design objectives:

- minimum number of gates,
- minimum number of inputs to a gate,
- minimum propagation time of a signal through the circuit, and
- minimum number of interconnections.

Quite often the application drives the implementation of the circuit making it difficult to satisfy all of these constraints all of the time. The classical method assumes that, given two circuits performing the same function, the one with the

fewer gates is preferable because it will cost less. This is not always the case with integrated circuits.

Mano [13] cites that it may be more economical to use as many of the gates from an already used package, even it the number of gates is increased. Moreover, some of the gate interconnections are internal to the chip and it is more economical to use as many of these interconnections as possible in order to reduce the number of wires between pins. With integrated circuits, it is not the count of gates that determines the cost, but the number and type of integrated circuits used and the number of external interconnections employed in implementation. Designing digital circuits is not a trivial task; many factors come into play. Along with the objectives previously mentioned, manually building the circuit on a circuit board is a tedious, time consuming task.

Joseph Greenfield [7] estimates that a circuit with 5 integrated circuits has "... no better than a 50% chance of working, due to wiring errors, when power is first applied." As the complexity of the circuit increases, the probability of the circuit working on the first try decreases rapidly. The process of locating and correcting the faults in a circuit is called "debugging."

Greenfield [7] states that the ability to debug a complex circuit is one criteria which separates the expert engineer from the novice engineer. To debug successfully, one must start by knowing exactly what the circuit is supposed to do and the value for each pin that is wired. By comparing the way the circuit actually works to the way it was designed should reveal the source of errors as either wiring or logic. Wiring errors consist of improperly connected gates, missing connections, and violations of fanout or race conditions. Logic errors, which are more complex to detect, require information about how the chip performs. By functionally decomposing the problem, a solution can be found; conversely, by assembling the partial solutions to a problem, the problem is resolved incrementally.

Expert systems thrive on problems that can be decomposed into subproblems. In general, an expert system has three main parts: stored knowledge (expertise), an inference engine, and a user interface. The stored knowledge is the fuel for the inference engine. As questions are posed by the user, the inference engine infers partial solutions from the stored knowledge. The sum of the partial solutions provides the completed answer.

An expert system's inference engine is the software that is executed to find the solution. The expertise, or knowledge, about the problem domain is represented in rules. Each rule has a premise and a conclusion. The premise is often referred to as the left-hand side (LHS) of the rule, and the conclusion is called the right-hand side (RHS). The LHS determines the applicability of the rule, and the RHS describes the action to be performed if the rule is applied [20].

The user interface is a critical part of any expert system. The interface should offer easy access to the stored data, rapid execution of problem resolution, and provide a friendly development environment for the user.

In the case of digital circuits, information needed to connect chips together could be viewed as knowledge for the expert system. Information such as the number of pins, value of each pin (input, output), type of chip, etc., are readily represented in a database. Evaluating the interconnections between one integrated circuit and another or to itself is beyond the scope of a database, but well within the power of an expert system.

Defining a broad-based debugging tool could prove to be ineffective as the design expands or unique instances occur. Defining a narrow debugging tool could limit the effectiveness of the debugger; therefore, a blend of debugging tools is needed. I propose that a two step approach could be used to debug a circuit.

Given a table of all the connections present in a digital circuit, the first step would use database functions to mark all the correct connections leaving the

incorrect, questionable, and interconnections among chips for the expert system. The list could then be reduced by removing the correct connections and passing a smaller list into the expert system for the second step of debugging. Errors would be identified by the expert system during the examination of the chip interconnections. This two step process would combine the speed of database functions with the power of expert systems. The result is a synergistic approach to debugging digital circuits.

Many of the current uses of expert systems (see Chapter 2) in computer aided design (CAD) work at the component level. Many assume that the circuit is wired properly and limit the aid to component layout or selection. Much of the research in CAD design is in the simulation of designed circuits. Also, considerable research has gone into routing algorithms for the pin connections. Using an expert system to debug a digital circuit at the gate level appears to be a novel idea.

The Air Force Institute of Technology (AFIT) School of Engineering has developed a prototype computer-aided design (CAD) workstation comprised of a simulator (LOGSIM) and a VAX based expert system (ICE). LOGSIM is a pin-level logic simulator which verifies circuit performance. Although ICE (Interconnect Expert) is used to identify wiring errors, ICE does not detect logic errors. Through the use of these two independent systems, a designed circuit can be tested without having to wire the actual components for verification [6].

The tools which are available are hosted on separate machines. The designer must learn how to operate each system in order to take advantage of the workstation. The expert system doesn't have a librarian or common user interface. It also lacks responsiveness and limits the number of integrated circuits which can be used. Also, the VAX is isolated from the lab.

LOGSIM and ICE need to be hosted on the same system to exploit the common circuit information and facilitate a more conducive design environment. LOGSIM and ICE operate on a limited set of problems without the use of a library function to add or modify existing integrated circuit (IC) descriptions. This creates a problem in

performing maintenance on the IC descriptions. Maintenance must be accomplished by explicitly changing the code, a very time consuming task.

Students attending AFIT School of Engineering are required to have a background in digital electronics. An important course is the initial course in discrete component design where fundamentals of circuit design and analysis are presented using transistor-to-transistor (TTL) devices. Although a CAD workstation exists, a complete system resident on one computer would assist the student in designing and testing TTL circuits. This workstation would allow the student to integrate different design tools into the early stages of the educational process. By exploiting this emerging technology, the student would be able to examine more complex problems [22].

1.2 Problem

This thesis addressed a number of problems in the CAD workstation. The first problem was hosting the expert system on a compatible machine with the logic simulator LOGSIM. Captain Wayne C. DeLoria's thesis [5] hosts LOGSIM on the workstation. Next, the issue of no user interface spawned a separate thesis effort to create a graphics environment capable of designing circuits and compatible with the expert system and simulator. Captain Charles A. Adam's thesis [1] addresses the graphics environment. Problems unique to the expert system included: lack of responsiveness, limited circuit size, user interface to the expert system, interface to the graphics package, and a librarian function to add and delete ICs. The goal of this thesis was to design, develop, and evaluate a microcomputer based expert system for digital logic troubleshooting. The primary research effort was to establish the merit of an expert system based on a limited domain problem. DBMS functions were used to facilitate the expert system interface design.

1.3 Scope

This thesis describes the design, construction, and testing of an expert system which detects wiring errors in digital design circuits constructed of 7400 series TTL ICs. The expert system includes a librarian utility which inserts/modifies the working set of IC components.

The issues of interfacing with the graphics package and the user were successfully resolved. The expert system, simulator, and graphics system are operational on the same microcomputer. The expert system provides explicit error detection message files and is able to handle large circuits. Expert system performance is evaluated and the extension of the knowledge domain, to include user generated design rules, is discussed.

1.4 Approach

The first step was examining the current literature regarding CAD development and related expert system developments. This examination was focused on the expert systems embedded in engineering workstations. Four software packages that could host the expert system were examined and evaluated. After the tool was selected, specific data attributes were defined and discussed with the individuals designing the simulator and graphic interface.

Implementation consisted of a combined database and expert system approach. A complete expert system was designed from the ground up and was evaluated with a suite of test cases. These test cases consisted of TTL circuits which were built and tested in EENG 450. Additional circuits were built to test for possible limitations on the size of the circuits. The expert system was modified and extended in an attempt to maximize the effectiveness of the tool as well as the quality of the 'advice.' A librarian feature was incorporated to allow for additions and modifications to the component list. The librarian was implemented in software and can be accessed through the user interface. Additionally, the expert system was integrated with the

other theses efforts to create a complete engineering workstation. Circuits designed by the students taking EENG 450 in Fall 1987 were tested and evaluated.

Features unique to the expert system in a stand-alone mode include: a natural language interface, the ability to create or edit graphics files, rule base querying on rule firings, trace record of rules fired, and the ability to decouple the control sequence of the expert system.

2. *Literature Review*

2.1 Analog and Digital Circuits

There are two types of circuits, analog and digital. This review will examine both cases; however, the primary use in industry is digital circuit analysis.

A research report by Wayne McNally [14] reviewed the various methods of working with analog circuits. This report presented a good overall approach to computer-aided design in an analog environment. McNally focused on the uses of SPICE (Simulation Program with Integrated Circuit Emphasis) developed by the University of California at Berkeley. SPICE is a public domain system that deals with floating voltage sources and inductances in an analog circuit. McNally reviewed systems available by eleven companies involved in computer-aided design. Each review provides a snapshot of the capabilities and limits of each design tool.

Several AFIT students [6] examined the effects of an expert system on digital circuits. Estes, Kroll, & Mraz designed and built a prototype in use at AFIT called InterConnect Expert (ICE). ICE is an expert system used to identify wiring errors in a digital circuit by using a rule-based system. Using ICE allows a user to check the wiring of a circuit without having to wire the actual components for verification. ICE is limited to thirty-two integrated circuits (ICs) for circuit design. ICE doesn't have a librarian feature for inserting or deleting ICs.

2.2 Electronic Workbench

The 21st Design Automation Conference hosted by IEEE in 1984 yielded some interesting work on current electronic workbench technology by Van E. Kelly [11], and William Birmingham & Daniel P. Siewiorek [2].

Van E. Kelly's work is an exploratory prototype called CRITTER and is used for critiquing digital circuits. It uses an expert system to determine functional

correctness, operating speed, timing correctness, and other circuit changes. CRITTER uses three steps to check the circuit. The first step transforms the user's information into the necessary specifications and signals. The second step transforms the data into a central database. Finally, CRITTER composes a cogent critique of whether or not the circuit will work. This type of analysis is necessary to insure circuit correctness before mass-production of a circuit board is initiated [11].

The work by Birmingham and Siewiorek addressed the issue of designing the entire board layout. MICON (Microprocessor Configurer) provides the designer with a workbench that takes a very high-level design description, and produces a network interconnection list. This list includes all the necessary hardware to design the circuit. The board is then ready for production. In the past, this type of layout was accomplished using manual techniques. The proposed method uses knowledge-based engineering techniques to drastically reduce the time required to produce a circuit board [2].

2.3 Knowledge-Base Systems

An IEEE conference held in 1985 provided a forum for presentations of some of the current issues in circuit design workstations. Two papers germane to knowledge-base techniques in circuit analysis were presented.

Leonard Moskowitz [18] of Allied Bendix Aerospace used a knowledge-base approach to design digital electronics. The system he built is called the Logic Designer's Apprentice (LDA). LDA prompts the user for the design parameters of the desired logic element, then identifies a circuit configuration and selects candidate components. LDA uses a unique method for selection or dismissal of candidate components and describes the reasoning behind each decision. The user can override the LDA and install a specific component. Once the circuit is complete, the design is displayed on a graphics terminal. In effect, the system emulates human methods of

component selection and interconnection. This system provides an apprentice type mode which assists the user in building the circuit.

Eric Sacher [21] provided a solid foundation for viewing software logic simulation at the microcomputer level. Logic simulation is the process of building a software model of a particular design, exercising the model, then evaluating the results. Sacher's model uses a series of twenty nodes to represent each IC. These nodes are used to represent the present and future states of the IC. The actual simulation of software uses a deductive method of determining the fault detection ability of the test program in diagnosis of the circuit. The simulator is controlled through a menu-driven user interface shell. The system is implemented on an IBM personal computer (PC).

2.4 Current Technology

Current technology that is commercially available is limited. Two of the more advanced systems are described below. HIWIRE is a schematic drawing system that adapts to an IBM PC configured with 320K RAM, two disk drives, parallel port printer, and DOS version 2.0 or later. It allows the design engineer to sketch or modify a design during all phases of conceptual development. It utilizes over 700 TTL components and costs $895 [10].

Hewlett Packard (HP) created the HP Electronic Design System. This system provides more than 3000 components and runs on HP workstations. This system has a capability for networking among a team of project engineers. Networking facilitates the use of shared computer resources. Each workstation can remotely access the central memory, transfer files, and share printers & plotters. The advanced user interface aids of pop-up menus, ICONS, and multiple windows help the user in design of the digital circuits [9].

2.5 Summary

Engineering workstations focus on simulation [2,11] as the cornerstone of development. Some systems implement the simulation tools through the use of knowledge-base systems [18,21]. Current technology systems [9,10] focus on the simulation aspects, but rely heavily on solid graphics and friendly user interfaces. These advances in software development address the issues of simulation and graphics; however, current systems neglect to employ expert systems in engineering workstations for digital circuit design.

3. Design Methodology

3.1 Design Approach

The expert system was designed using the four stage expert system development methodology proposed by Citrenbaum, Geissman, and Schultz [4]. Their design methodology is similar to the Harmon and King development scheme [8] and Metzger's rapid prototyping approach [15]. All of these approaches begin with a problem determination stage, develop a prototype, extend the prototype, and conclude with an integration and maintenance phase. Essentially, each system develops a prototype in the second stage and then extends the prototype in the third stage. There is very little difference between each method, Harmon and King use six stages compared to the four stages of the other systems.

The Citrenbaum method uses four stages: problem determination and specification, initial prototype, expanded prototype, and delivery system. Each stage stresses different features of the expert system. An overview of the Citrenbaum method is presented in Figure 1.

The problem determination and specification stage corresponds to the requirements analysis stage in a conventional software development project. The major objective is to insure that the project will satisfy a real need and be technically feasible. The main steps are to determine whether an expert system approach is suitable for the problem, identify a subset of the problem for the prototype, and identify the knowledge requirements so that the program structures and tools can be evaluated.

The main objective of the initial prototype is to rapidly demonstrate the technical feasibility of the expert system. The initial prototype represents a small version of the completed system. It is designed to test assumptions about how to encode

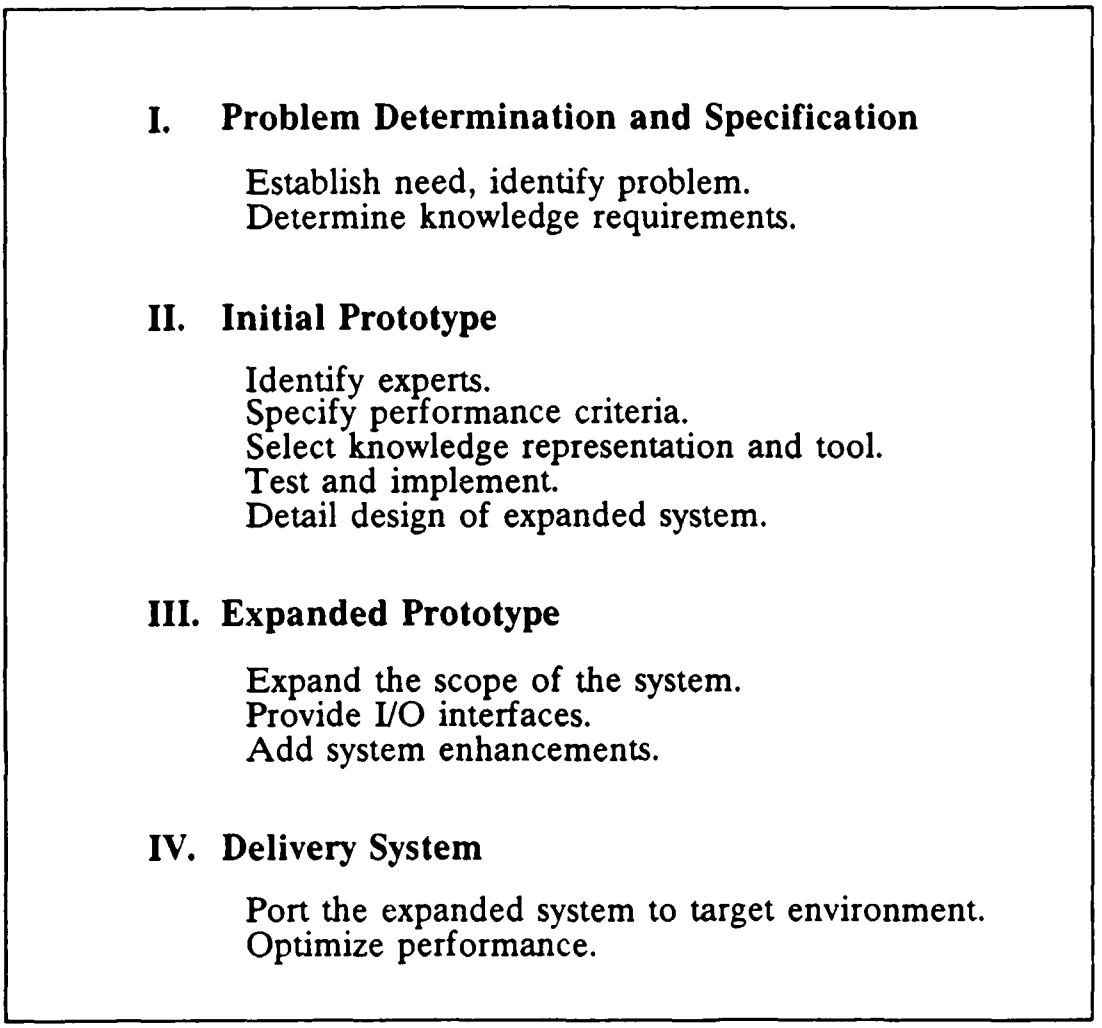


Figure 1. Citrenbaum Expert System Development Methodology

the facts, relationships, and inference strategies. Additional activities in the initial prototyping phase include:

- identifying the experts for the system,
- learning about the problem,
- specifying performance criteria,
- selecting an expert system building tool,
- developing an initial implementation,
- testing the implementation with test cases, and
- a detailed design for the complete expert system.

Once the initial prototype demonstrates that the expert system is feasible, the expanded prototype can be developed. Development should include: expanding the scope of the system, provide I/O interfaces, monitoring system performance, and adding the bells and whistles. The major development problems in this stage tend to be technical in nature. The expert system tool plays a major role in handling technical problems.

The last stage, delivery system, has two main goals: port the expanded prototype system to the target environment, and meeting the software engineering issues of the project. These issues include performance, integration with other systems, portability, modularity, debugging support, and adherence to design standards. The user interface should meet the customized needs and documentation for system maintenance must accompany the final product.

These four stages provided an excellent baseline for the development of the expert system for discrete component digital circuit design. The remainder of this chapter will detail how the expert system was designed using Citrenbaum's four stages.

3.2 Problem Determination and Specification

Chapter 1 outlined many of the pitfalls associated with digital circuit design. Clearly, there must be a less costly and more efficient way to design digital circuits than building them by hand on a circuit board. This thesis addresses the expert system part of an engineering workstation that will allow a designer to engineer digital circuits using a computer with an oncall design assistant.

The expert system was implemented on a Zenith 248 microcomputer; an IBM compatible, DOS based system. The expert system interfaces with the other theses efforts [1,5] which comprise the engineering workstation. Therefore, it was necessary that the expert system building tool be DOS based and operate within the constraints of the Zenith microcomputer.

A key factor in building an expert system is the identification of knowledge required for the system. Knowledge needed to design and debug digital circuits was readily available since digital techniques are well documented in textbooks. The transistor-to-transistor (TTL) chips used by the expert system can be found in the manufacturers handbook [23].

The expert system was required to operate in a stand-alone mode, as well as integrated with the workstation. Therefore, a database of TTL chips was built for the expert system (Appendix B.) The initial workstation library consisted of 32 different chips (Appendix A.) Armed with the basics of an expert system, the prototype was constructed.

3.3 Initial Prototype

Before the design of an initial prototype could begin, several issues were resolved. It was necessary to identify the experts for the system, system performance criteria, and evaluate and select an expert system building tool.

The expert knowledge or expertise for the expert system was obtained from Mano[13] and Greenfield [7]. These textbooks provided the elementary rules for digital circuit design. The Texas Instrument TTL Handbook [23] was used for the chip information needed for the database.

The system performance criteria of the expert system consisted of: the identification of wrong connections in the circuit, the identification of missing connections, and any violations of digital design principles such as race conditions or fanout. The system had to be able to evaluate the maximum size circuit built by the graphics package, a circuit of 8 - 10 ICs was used. Finally, the expert system was required to return an echo copy of the ASCII input file along with error messages for wrong or missing connections.

A key step in prototype development is the tool selection for the expert system. There are a wide range of expert system building tools. Four commercially available software packages that could host the expert system were examined. They were: C, OPS5, OPS83, and Guru. A small project was implemented in each candidate language/tool so that the different attributes of C, OPS5, OPS83, and Guru could be examined. A summary of each candidate has been provided. The evaluation and final selection of the design tool follows these summaries.

3.3.1 C Programming Language. C is a general purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators. It was originally designed for and implemented on the UNIX operating system. C is not tied to any particular hardware or system; however, it is easy to write programs that will run without change on any machine that supports C [12].

C is a relatively low-level language. This means it deals with the same sort of objects that most computers do, namely characters, numbers, and addresses. C provides no operations to deal directly with composite objects such as character

strings, sets, lists, or arrays considered as a whole. Finally, C provides no input-output facilities: there are no READ or WRITE statements, and no wired-in file access methods. All of these higher-level mechanisms must be provided by explicitly-called functions.

C offers only straightforward, single-thread control flow constructs: tests, loops, groupings, and subprograms, but not multiprogramming, parallel operations, synchronization, or coroutines. C does not offer a predefined rule set or control strategy. All the basic components of the expert system must be written.

3.3.2 OPS5 Programming Language. OPS5 is a general purpose production-system language. A program in OPS5 consists of a declaration section, describing the data objects of the program, and a production section containing the rules. During program execution, the rules are kept in production memory and the data is kept in working memory.

The inference engine consists of three steps. The steps are: match-rules, select-rules, and execute-rules. Match-rules matches condition elements against working memory, select-rules chooses one dominant rule, and execute-rules fires the rule by executing the actions of the RHS.

OPS5 supports forward chaining which is based on executing each rule in a forward direction, first looking at the premise, the conclusion is ignored until the premise is true. When the premise is true, the rule is fired. If the premise is false, the rule is not fired and another one is considered. The firing of a rule could change a variable in the premise of another rule causing it to be eligible, this is called forward chaining. This process continues until the solution is found–the goal is met [17].

The language defined data types can be primitive or compound. Primitive types consist of scalar types–numbers and symbolic atoms. Compound types are an element class with components called attributes. Since OPS5 is not a strongly typed

language, there is no mechanism for declaring a variable or an attribute name to be a particular type or to define subtypes of the basic primitive type [3].

OPS5 memory is defined as working or production memory. Working memory is where the data is collected and maintained. Elements of working memory are created by a make command, this command must be preceded by both declarations and a rule definition. Each element is associated with a time tag or recency attribute. Production memory consists of LHS (condition elements) and RHS (actions). Every rule of the production system must be entered into the production memory. The LHS is basically defined as a sequence of condition elements, whereas the RHS is composed of a sequence of actions–there are twelve predefined action types in OPS5. The description of the twelve actions are omitted here, but are described in [3].

The OPS5 inference engine is a three-stage process of matching, selecting, and executing rules. This process is the basis of control in a production-system language. The inference engine cycles over these three states during execution. Exiting occurs after the match state, because there are no more rule instantiations in the conflict set or because of an explicit halt.

Matching builds an instantiation for every set of elements that match with consistent bindings, these are placed in the conflict set for a given cycle. Selecting uses two conflict resolutions to select an instantiation from the conflict set–LEX & MEA. LEX is the simpler of the two, MEA places extra emphasis on the recency of a working memory element that matches the first condition element of the rule, thus making it easy to implement depth-first search using goal elements as the first condition. LEX strategy gives no special preference to the first condition element. Both make several passes over the conflict set to select the best rule to fire. Executing carries out the action of the rule.

Efficiency is an important consideration in any production system. In OPS5 the matching process is augmented by the Rete algorithm. The Rete algorithm is efficient because it doesn't match all elements on each cycle, it shares similar tests

in different rules, and it recomputes whether or not combinations of matches are consistent only when necessary.

The authors [3] point out that the main advantages of OPS5 are its stability and efficiency. It has been used to implement a number of moderately large expert systems such as R1 and XSEL. Disadvantages are its lack of well-developed user interfaces, special editing or explanation facilities, and aids for maintaining test case libraries. Features such as backward chaining have to be programmed explicitly when they are desired. However, recent implementations have corrected some of these disadvantages.

3.3.3 OPS83 Programming Language. OPS83 is a programming language used for expert system development. OPS83 is a descendant of OPS4 and OPS5. Two main differences between OPS83 and its predecessors are:

1. OPS83 supports imperative programming languages such as C and PASCAL, as well as rule based systems like OPS5. This is an important feature because it allows for a mixture of rule based and imperative programming, depending on the task [19].
2. OPS83 could be considered a lower-level programming language than OPS4 and OPS5. OPS83 provides flexibility on how rules can be fired, the interpreter doesn't have a set control cycle. This allows the user to tune the rule set to a specific application.

During program execution, the rule interpreter performs a recognize-act cycle. This cycle consists of three stages. The first step, Match, evaluates the LHS of the rule, checking for satisfied conditions. The second step, Conflict Resolution, searches for a rule with a satisfied LHS. If one is found, the third step, Act, performs the actions of the RHS of the selected rule. This cycle continues until the goal is reached or halt is invoked. Halt is invoked if no rules have a satisfied LHS.

The imperative component of OPS83 is supported by constants, expressions, executable statements, input and output commands, modules, and [illegible]. Constants and expressions consist of five kinds of constants such as integer numbers, logicals, characters and symbols. Expressions consist of arithmetic, relational, comparative or logical.

Executable statements support assignment statements, read, write, and [illegible] statements. A cast statement allows for the assignment of a variable to a different variable type, such as assigning an integer to a floating variable or vice versa. Compound statements, if, while, and for statements are also supported.

Input and output are handled in one standard file format designated by OPS83. The file consists of character sequences separated by new line characters. Binary strings are not used, rather each object in the file are character strings. The actual manipulation and storage is transparent to the user, the key point is that all data is viewed as character sequences.

Modules encompass the declaration of variables, functions, and procedures. Both local and global variables can be used. A function or procedure can be declared as simple, which means it produces no side effects. Functions or procedures which produce side effects can not be called from the LHS of a rule. The compiler is responsible for checking functions. A function or procedure is considered simple if it does not have global references, read or write statements, modify or move statements, and calls only simple functions or procedures. OPS83 does support recursion.

OPS83 supports many data types such as arrays, records, extended records, and derived types. Arrays and records are declared in the program. Extended records can contain records or arrays as part of the record composition. Nesting of records or arrays is permitted in the same record. Derived types are named types that are defined as equivalent to some existing type. A derived type inherits all the properties of the parent type.

OPS83 uses two levels of working memory. The low level is used for utility functions such as printing or clearing memory. The high level is used by the rule set during execution. It's necessary to make this distinction because OPS83 allows the user to define the interpreter cycle, this cycle affects both portions of memory.

Implementing the recognize-act cycle, the execution of the rule set, is a unique feature of OPS83. The conflict set is made up of instantiations (a rule and its list of working memory elements that satisfy its LHS) that exist in the current system. Every time the memory is changed, the system examines the LHS of the rules and determines if changes are to be made, these changes are updates to the conflict set. OPS83 programs do not directly manipulate the conflict set; rather by using predefined functions, the user is able to access the information in the conflict set.

Functions are also used for conflict resolution. Similar to the MEA strategy of OPS5, OPS83 offers a function named select. The select procedure has two main loops. The first loop searches for unexecuted instantiations, the second loop searches for an instantiation in the following manner:

- After examining the unexecuted instantiations, the instantiation with the largest working memory index is selected. From these, the instantiations with the greatest number of patterns are identified. In the final pass, the one that entered the conflict set most recently is selected.
- OPS83 does offer a trace capability for all the rules which fired. Additional information on partial firings, working memory, and the conflict set can be retrieved. OPS83 does not offer many prewritten procedures, the user must author these routines along with the input and output routines.
- OPS83 does not allow backward chaining, but does allow for multiple forward chaining search strategies. OPS83 does interface with C or other C based languages.

3.3.4 Guru Programming Tool. Guru is an expert system building tool designed to integrate a host of unique software features into one building tool [17]. Guru offers four ways for the user to interface with the system: Menu's, Command Language, Customized Interfaces, and Natural Language. Menu guidance is helpful for the first time or casual user. All the Guru features can be accessed through the menus. A more experienced user might prefer the Command Language interface. The commands are english-like and offer over 500 context sensitive help screens. The commands are helpful in designing the Customized Interfaces. The Customized Interfaces support windows, menus, and forms. The Natural Language interface is used within a Guru session. Guru prompts for additional clarification if necessary.

Guru is comprised of more than a dozen components. All of these components are accessible through the menus or command interface. Listed below is a brief description of each component.

- Expert System Construction: The construction consists of building a rule set. Guru does not limit the construction of a rule set to rules, a myriad of options are available. Extensive initialization and completion sequences are allowed along with security controls and visual action controls.
- Expert System Consultation: Guru supports both forward and backward chaining. Guru is capable of reasoning with certainty factors in any of sixteen ways, and allows for fifty rule selection strategies. Also, Guru has an explanation capability feature built-in along with a trace file of rules fired. The Guru forward chaining strategy executes the same as described earlier in the OPS5 review. Backward chaining is based on considering rules in the reverse direction–looking at the conclusion, rather than the premise. The inference engine identifies rules whose conclusion could reach the goal. It then determines if the premise is true. If it is, then the rule can be fired to produce a solution. If it is not true, or perhaps unknown, the inference engine considers these variables to be new subprograms. It tries to solve each of these subprograms by

backward chaining with other rules. This process continues until the goal is found or the problem is found unsolvable [16].

- Relational Data Management: Tables are available with a wide range of features such as sorting, selecting, deleting, modifying, security levels, data masking, and indexing. Guru also uses virtual fields for dynamic processing.
- Ad Hoc Inquiries: The query syntax is similar to IBMs SQL/DS mainframe relational database system. Guru extends this feature to include data retrieval from multiple tables, wildcard string and symbol processing, multi-level control breaks, and dynamic sorting and editing.
- Statistical Analysis: Full statistics are available on any numeric or integer field in a table. The list includes min, max, sum, count, mean, standard deviation and variance. These can be applied to several tables if desired.
- Calculations: Guru supports an extensive set of functions to work with numeric or string data.
- Spreadsheet Analysis: Along with the normal spreadsheet capabilities, Guru offers a host of options that can be applied down to the individual cell. These options include blinking, bell, reverse video, invisibility, color changes and presentation of the information in the cell. All cell values are accessible throughout any level of processing.
- Screen Forms Management: Screens can be built through menus and edited in the text mode. All the features associated with tables and spreadsheets apply to forms, forms are used just like procedure calls.
- Printed Forms Management: Pre-printed forms can be used because Guru can mask the output into any style or format. Printed forms can be routed to disk files for printing at a later time.
- Procedural Modeling: All the procedural constructs are supported, such as if-then-else, while-do, and case. Procedures can have up to 26 parameters. These

parameters can be table values, spreadsheet information, or global variable information.

- Text Processing: Guru has horizontal and vertical scrolling, over 20 control functions, 40 format controls, word wrap, block processing, and a full screen editor. Any of the control/command functions can be incorporated into the text and the results incorporated into the final report.
- Business Graphics: Graphs, plots, and table information can be generated and displayed using headers, footers, and numerous report details.
- Remote Communications: Guru supports direct communication with other computers, file transfer, and terminal emulation. Over 20 communications environment controls are furnished to tune the use of the communication package.
- Postrelational Data Base Management: An extensive list of postrelational options is available in Chapter 18 of the Guru Reference Manual [17].

These features were extracted from the Guru Reference Manual. A more extensive description and examples of each feature is available in reference [17]. The integration of these components is based on the principle of synergy–the total effect is greater than the sum of the individual parts. Guru eliminates all conventional barriers of switching back and forth between processes or having to complete one process before starting another. All are blended into one environment and are accessible at any time. For instance, the premise of any expert system rule can directly reference a table, program variable, or array. One expert system could call another, or prompt the user, by using a form, for additional information.

The selection of an expert system building tool is the last step before the design of the prototype can take place. In a book by Donald Waterman [24], six basic questions are provided for selecting a tool. These questions were used as a guideline for picking the expert system tool and will reference the information provided on C, OPS5, OPS83, and Guru.

Question 1. Does the tool provide the development team with the power and sophistication they need? All the languages have the power; however, C, OPS5, and OPS83 definitely lack the sophistication to provide a solid user interface and data structures necessary to handle the data and output requirements. Guru has numerous predefined functions and procedures to implement any requirement. Any data structure above a procedure call would have to be authored if C, OPS5, or OPS83 were used.

Question 2. Are the tool's support facilities adequate considering the time frame for development? Guru is the only tool which qualifies under this constraint. The power of the menu's and command environments reduce the workload to a composite of keystrokes versus manipulation of elementary code if any of the other languages/tools were used.

Question 3. Is the tool reliable? All of the tools are available, operational, and complete. Current licenses and documentation are available for each package.

Question 4. Is the tool maintained? Yes, all of the tools are covered by the host department at AFIT.

Question 5. Does the tool have the features suggested by the needs of the problem? Recall, a tool is needed that can handle the many interface requirements between the expert system and graphics, expert system and database, and expert system and user. Also, the ability to change the library of ICs and the ability to handle large circuits should be examined. All tools meet this criteria; however, C, OPS5, and OPS83 would require extensive coding to create the environments to handle the interface requirements. Guru offers a support environment that is conducive to manipulating large amounts of information through various environments. In some cases, such as entering the input data, Guru uses only one line of code to load the entire contents of an ASCII file into a table. C, OPS5, and OPS83 would require considerable coding to accomplish the same task.

Question 6. Does the tool have the features suggested by the needs of the application? The application suggests a fast, accurate, user friendly system. All the packages would deliver an accurate program. Using a low level language such as C, OPS5, or OPS83 could provide a faster processing time; however, Guru is clearly the only choice for the user friendly requirements.

The tool selection boils down to considering the following facts:

- the descriptions of the candidate languages/tools,
- the answers to the tool selection questions,
- the lack of higher order constructs for C, OPS5, and OPS83, and
- the vast routines and options that Guru has to offer.

All things considered, Guru was picked as the tool to use for this thesis. Prototype development began with a firm decision to use the Guru development environment. The main objective of the initial prototype was to rapidly demonstrate the technical feasibility of the expert system using Guru Version 1.0.

3.3.5 Graphics Interface. The expert system relies heavily on the interface to the graphics package. This interface is depicted in Figure 2. The graphics package places an ASCII file in the DOS environment, then envokes Guru. Guru automatically executes the expert system by invoking a startup file which acts as a mainline program. The expert system uses the input file to start the processing and generates an output file, containing any circuit errors, upon completion. The output file signals the end of execution and control is return to DOS, which invokes the graphics system.

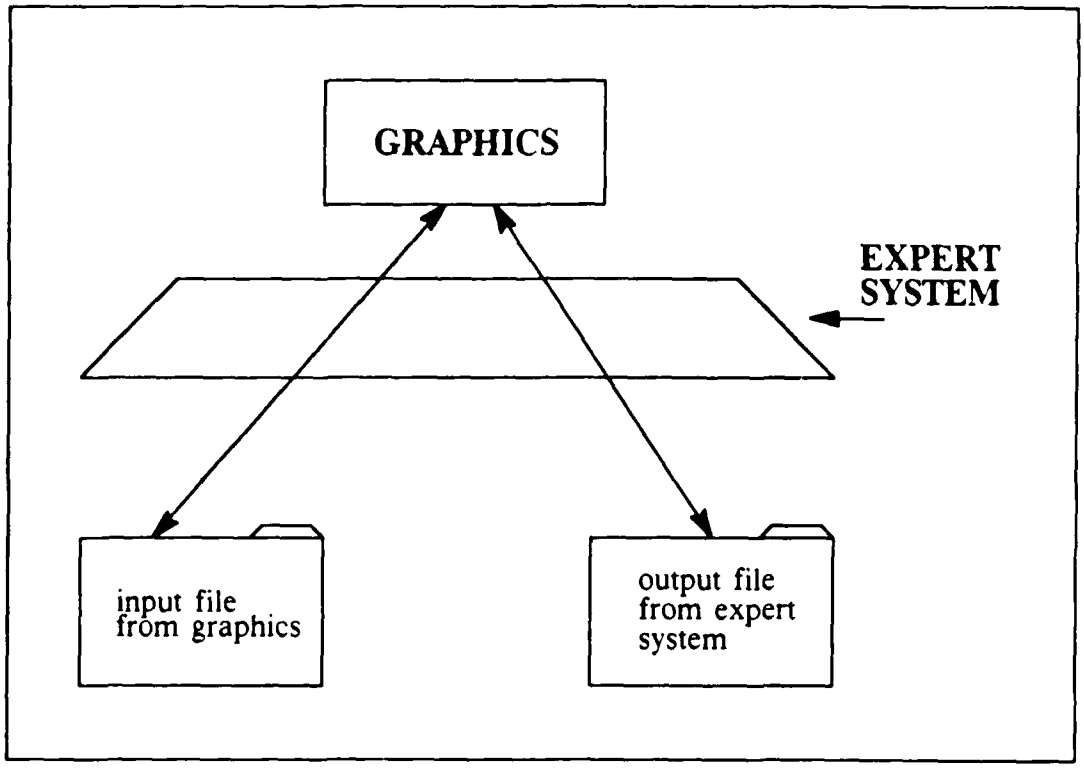


Figure 2. Graphics Interface

3.3.6 Prototype Schematic. First, it was necessary to define the interface to the graphics package. Second, the expert system requirements were analyzed. Finally, an overview of the system was created. Figure 3 depicts an abstract view of the expert system. The solid lines partition the prototype into the input phase, processing phase, and output phase. Each of these phases were decomposed and implemented in Guru.

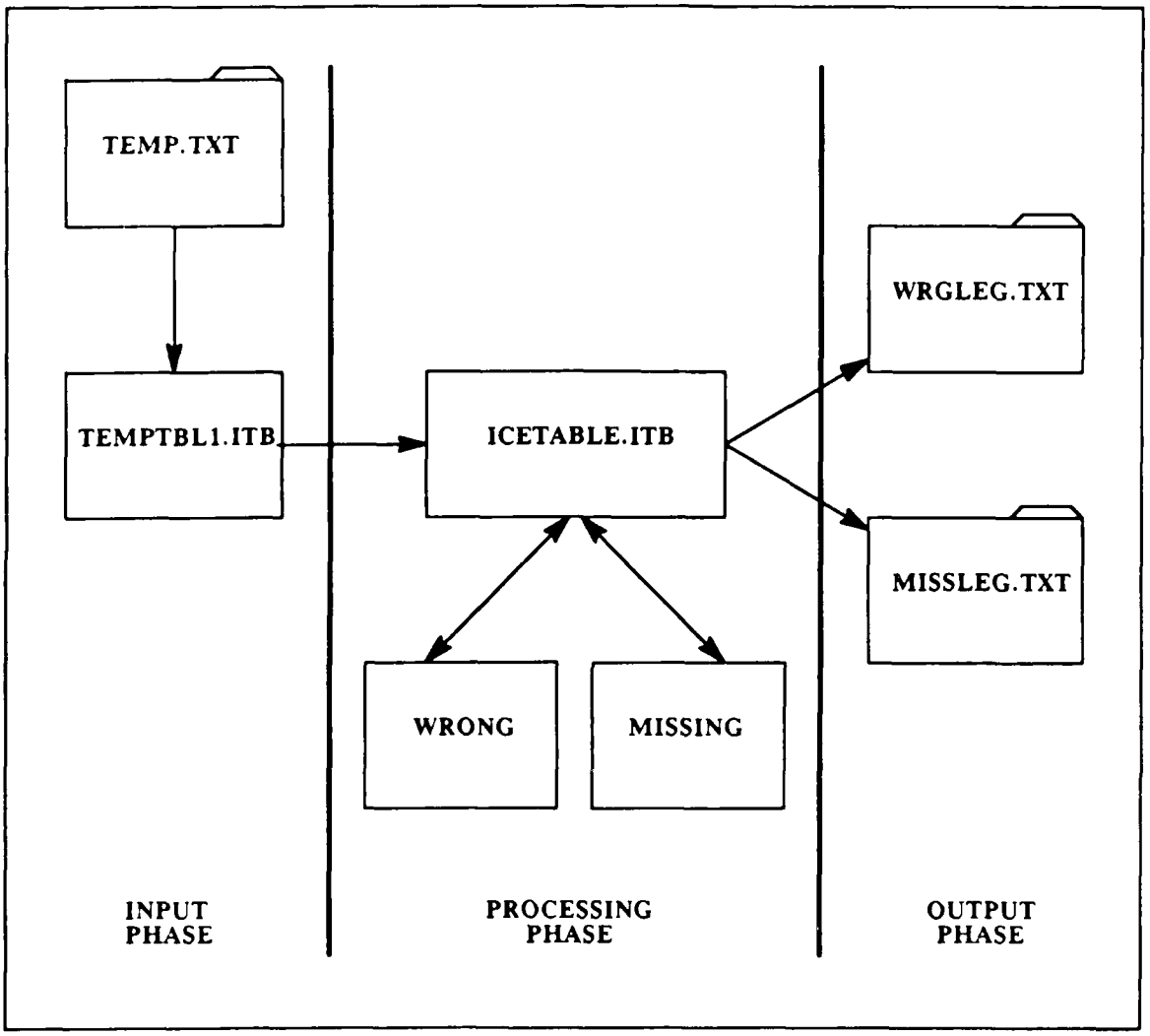


Figure 3. Prototype System Layout

3.3.7 Input Phase. Figure 4 is an example of a simple TTL circuit drawn by the graphics package. Each chip has a unique label, such as T001 or T002. The type of TTL is also given, in this case a 74116 and a 7400. The TTL pin numbers have been added for clarity. Note the two types of connections: TTL to TTL and external source to TTL. The entire circuit is built using these two types of connections. These connections are listed in a file, TEMP.TXT, and given to the expert system as input data. This file is converted into a Guru table in the input sequence.

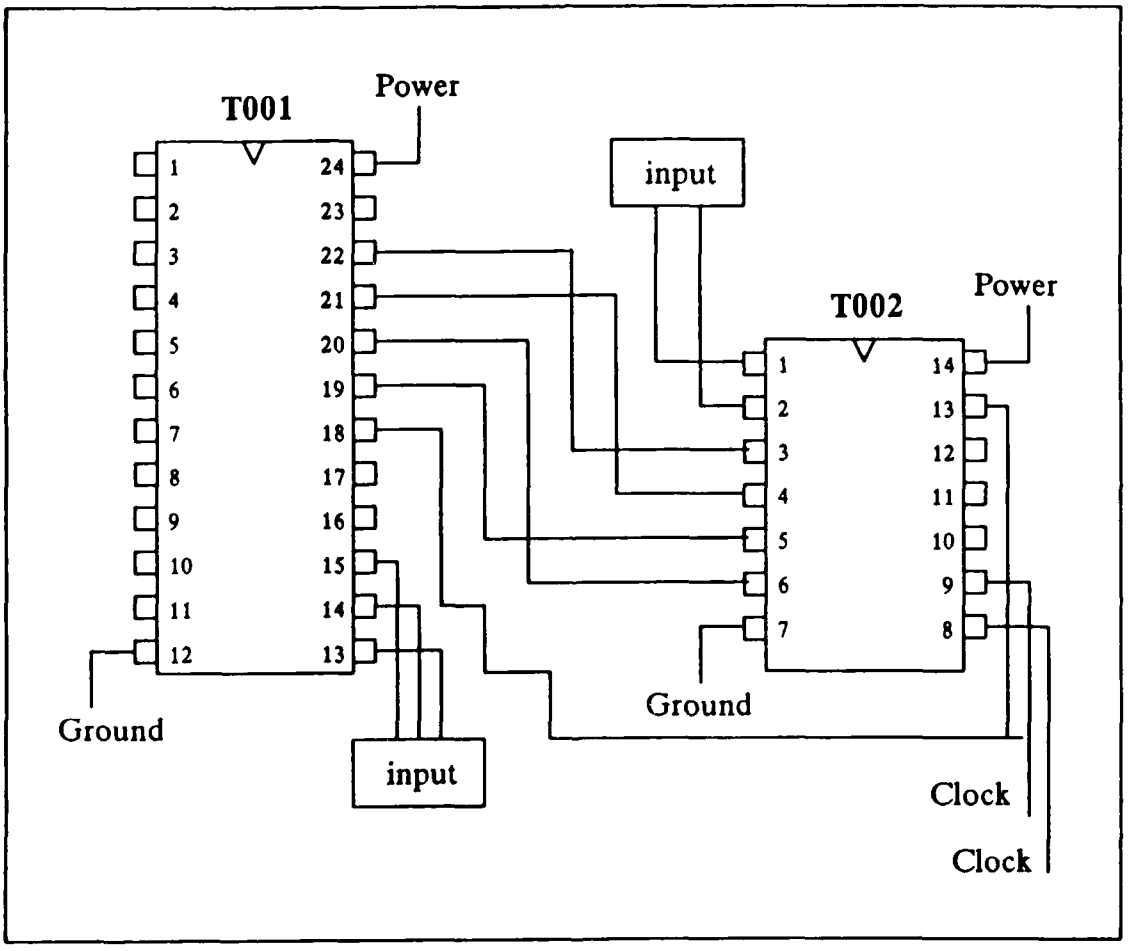


Figure 4. TTL Circuit

3.3.8 Processing Phase. The processing phase begins by creating a larger table called ICETABLE.ITB. ICETABLE holds all the initial input data, temporary variables, error comments, and database information. ICETABLE extracts circuit information from TEMPTBL1.ITB and creates a row in ICETABLE for each pin connection. Figure 5 shows ICETABLE after an input file (the circuit in Figure 4) is loaded. After assembling the pin connections, ICETABLE consults the integrated circuit (IC) database to determine what value should be present on the pin. The database is resident in a table called ICTABLE which contains 32 different ICs. Figure 6 shows a composite of ICTABLE, Appendix B contains an entire listing of the contents of ICTABLE.

Retrieval from ICTABLE is accomplished in the following manner: Figure 4, shows package T002 as a 7400 with a value on pin 14. The database shows pin 14 of a 7400 should be connected to power, so power is loaded into ICETABLE. Each pin connection is evaluated for what value should be present and the value loaded into ICETABLE for processing at a later time. This processing feature takes advantage of the predefined compare routines resident in the database package.

ICETABLE.ITB

```
T001.....05.....b...        : I000.....xy.....
I000.....mv.....            : T001.....01.....a
P000.....                   : .....
C000.....                   : T002.....01.....a
T002.....09.....b...        : G000.....
T001.....07.....chip        : G000.....
```

Figure 5. ICETABLE Example

ICTABLE.ITB

```
007400      01    a
007400      02    a
007400      03    a
     .....
007400      14    chip
     .....
     .....
     .....
074116      01
074116      02
074116      03
     .....
     .....
074116      24
     .....
```

Figure 6. ICTABLE Example

Once ICETABLE is loaded with the input file and database information, a list of the pin connections is created. This list is sorted by chip and pin number then passed to the expert system, along with the ICETABLE, for processing. The list of chips is called ICLIST, Figure 7 depicts ICLIST. The expert system decomposes the circuit chip by chip. The premise follows: in order to verify that a circuit is correct, the expert system must verify each chip. In order to verify that each chip is correct, the expert system must verify each gate. Once each gate of a chip is verified, the next chip is processed. This recursion continues until all the chips are evaluated. Figure 8 shows the circuit decomposition.

The expert system uses ICETABLE and ICLIST to determine two things: wrong legs and missing legs. Wrong legs detection is a mixture of database functions and expert system commands. Missing legs executes database functions only. The combination of wrong legs and missing legs comprise the debugging feature of the expert system.

3.3.8.1 Wrong Legs Wrong legs are determined by a two stage process. Stage 1 uses database commands to evaluate the contents of ICETABLE. For instance, according to the database, pin 14 of T002 should be connected to power. The circuit information, input from the graphics, shows pin 14 of T002 connected to power. Therefore, the connection is correct and no further processing is required on this row in ICETABLE. The first stage only evaluates external connections to TTLs or TTLs connected to external connections. Interconnections between TTLs are evaluated by the expert system. If an error occurs such as pin 14 of T002 connected to ground, it is flagged and processed in Stage 2 along with all the TTL to TTL connections.

Stage 2 handles all the interconnections between TTLs as well as all the errors caught in Stage 1. A rule set called ICERULE.RSS was used to categorize the error and generate the error message. Since this is the prototype, the rule set was not

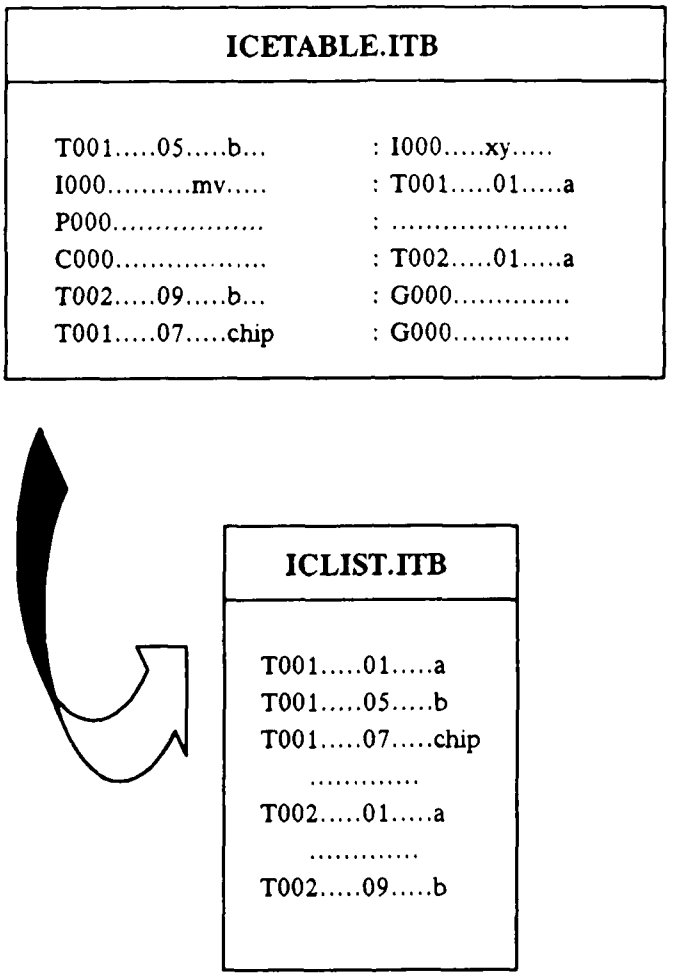


Figure 7. ICLIST Example

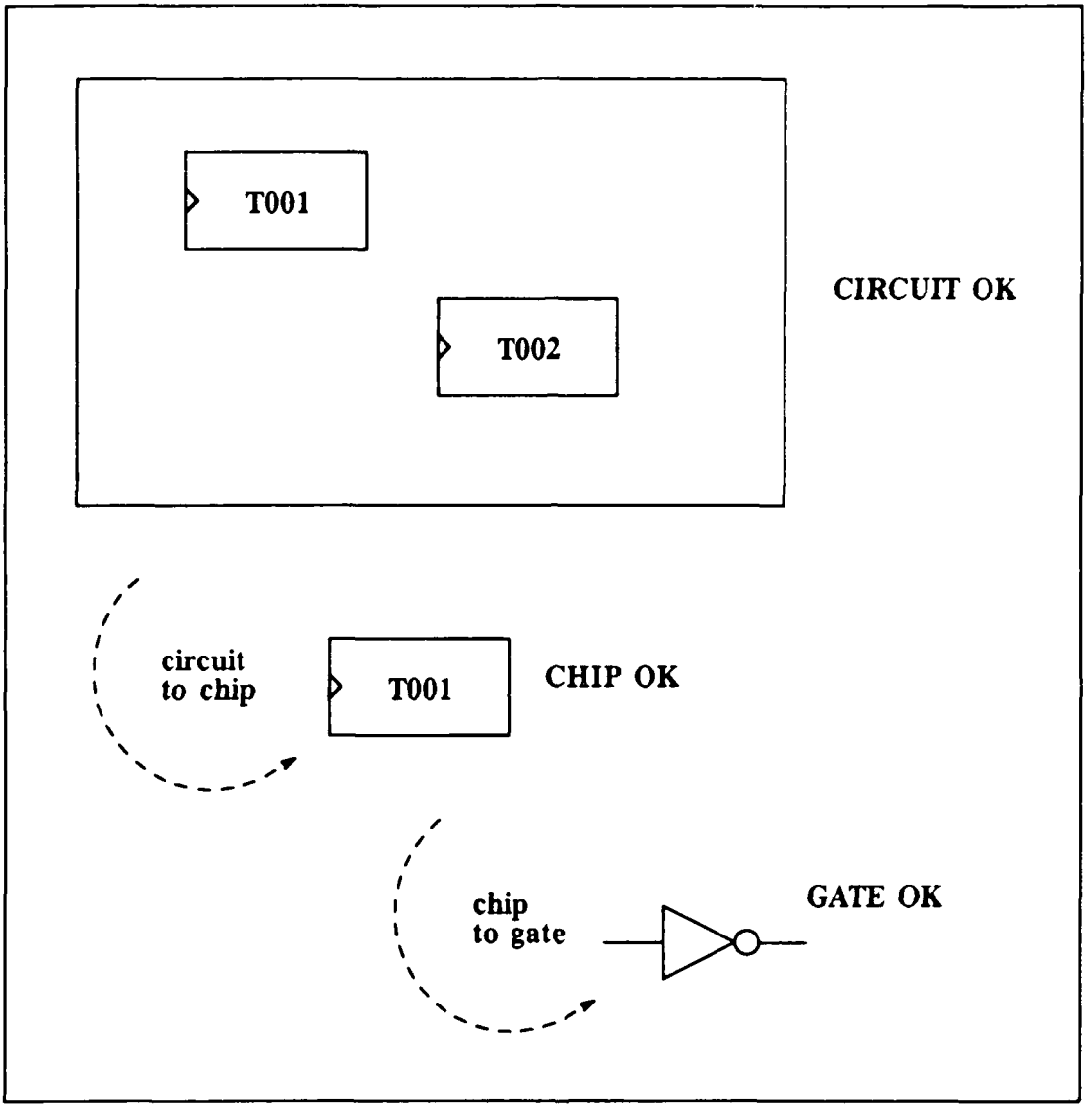


Figure 8. Circuit Decomposition

completed. A small set of rules were used to capture the occurrence of a TTL to TTL connection, and messages were sent to the error file. The main objective was to test the combination of using database features in concert with rules to evaluate all the rows in the table. Once all the rows were evaluated control was passed to the 'missing' function.

3.3.8.2 Missing Legs. Missing uses database operations to evaluate the circuit for missing legs. Since a circuit consists of one or more chips, it is necessary to isolate a chip and then inspect it's connections. Furthermore, a chip consists of multiple gates; therefore, it is necessary to inspect each gate. In order to verify existing connections and identify missing ones, it is necessary to obtain a 'master' copy of what connections are legal from the database. ICTABLE is queried for an IC, when it is found, a table called ICMASTER is created. ICMASTER was created to handle the many 'database' requests concerning chip specifics. If these requests were made against the large ICTABLE, processing time would increase. Using ICMASTER reduces processing time because it contains only information germane to the chip being processed. Figure 9 shows ICMASTER as a product of ICTABLE.

Once the ICMASTER is obtained, the first IC is extracted from the ICLIST and put into ICCHIP. ICCHIP is broken down by gates and one gate at a time is processed looking for missing legs. Figure 10 shows how ICLIST passes information to ICCHIP and then down to ICGATE.

ICGATE has a list of all the gate connections present in the circuit, but it is necessary to have a list of the legs which must be present. This list is obtained from the ICMASTER. ICMASTER is a list of all the pins associated with the chip being evaluated. Figure 11 shows how GTMASTER is built.

GTMASTER is a list of all the pins associated with a specific gate. A one-to-one match is made between ICGATE and GTMASTER, the unmatched legs in

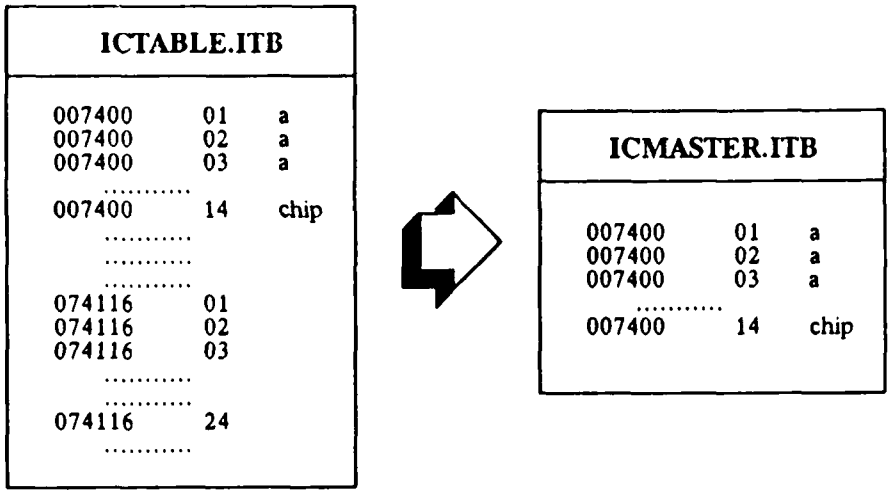


Figure 9. ICMASTER Example

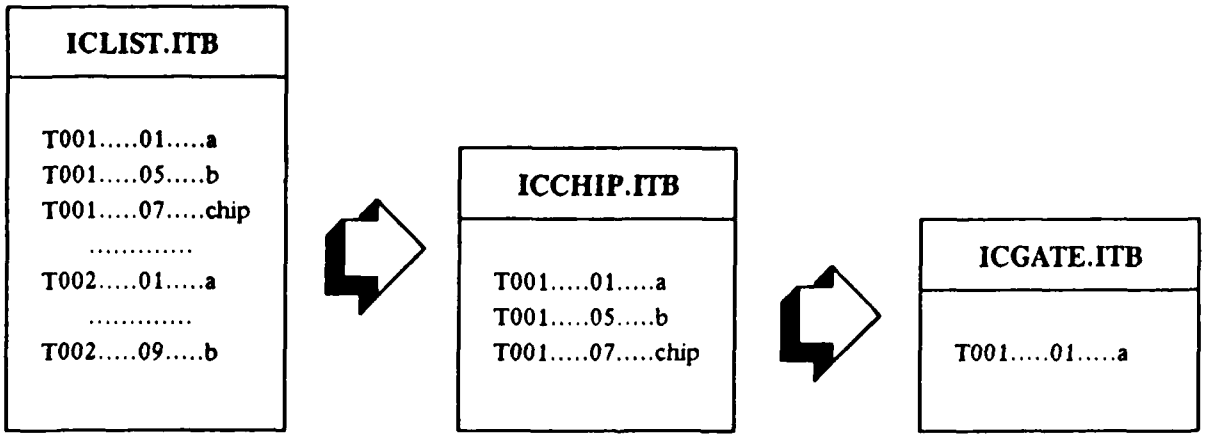


Figure 10. Decomposition of ICLIST

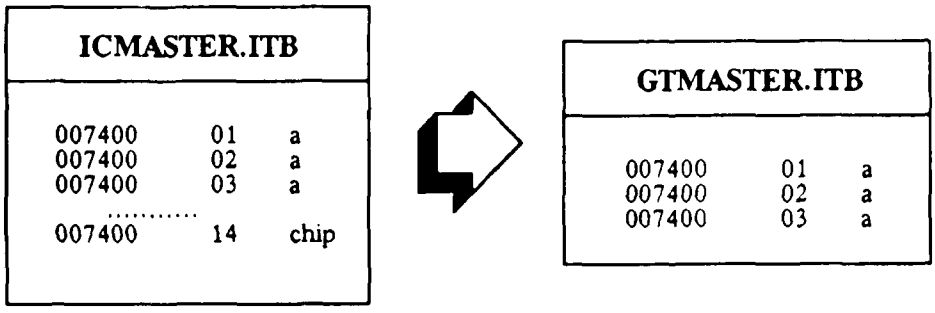


Figure 11. GTMASTER Example

GTMASTER are 'missing legs.' These legs are written to a file and accumulated during circuit processing. Once all the chips are processed for wrong or missing legs, control is passed to the output phase.

3.3.9 Output Phase. The output phase is the interface back to graphics. Once the output files are generated by the expert system, the output phase generates the necessary ASCII files for the graphics system. Control is then returned to DOS for further processing. In the prototype, two files were created, one for missing legs and one for wrong legs. The tables were created by loading the tables into a dynamic array and then converting the data into ASCII files.

The interface requirement specified that the output format should mirror the input format with the addition of a 'y' for the connections that were wrong. Dynamic arrays were used to pack and unpack the files with the proper syntax and to create the ASCII files.

3.3.10 Prototype Evaluation. All phases of the prototype were implemented and operational in Guru. Several design problems did occur, the main problems were related to the memory constraints of the Z-248 microcomputer.

Guru requires 448K to execute and additional space is requested in 48K continuous bytes. In the Z-248 the working memory is limited to approximately 580K. During the processing of the expert system, Guru would request space and none would be available. As a result, a memory error would abort execution. In another case, the arrays needed to build the output files could not be dimensioned because memory was already saturated, once again corrupting the execution. Other instances included lack of memory to build the tables, or space to sort the large tables. There were several possible solutions.

The first solution was to limit the size of the circuit the expert system could handle. This wasn't the desired course of action, the expert system should be able to analyze any size circuit the graphics could build. The second solution was to combine some of the tables to cut down on the overhead. This approach was feasible and in several instances, some of the temporary tables could be eliminated.

The prototype was tested using simple and complex circuits, in all cases the prototype found all the wrong and missing legs. Aside from the memory errors, Guru was able to handle all the desired features of the expert system. In order to extend the prototype, the memory problems must be addressed, the rule set must be redefined, and the interfaces must be 'cleaned up.'

The biggest problem was handling the memory errors. Changes in the input phase was a step in the right direction. Instead of loading the input file into a temporary table, data from TEMP.TXT was loaded directly into ICETABLE. This change saved space and also cut down some of the processing time. Elimination of arrays needed for the output sequence eliminated the possibility of fault errors in trying to dimension the arrays. Since the output is ASCII text, dynamic tables were used to create the output files. This modification eliminated the possibility of array memory errors. Other table manipulations were modified to minimize the activity with working memory. Figure 12 reflects the modified prototype layout to be used in the extended prototype.

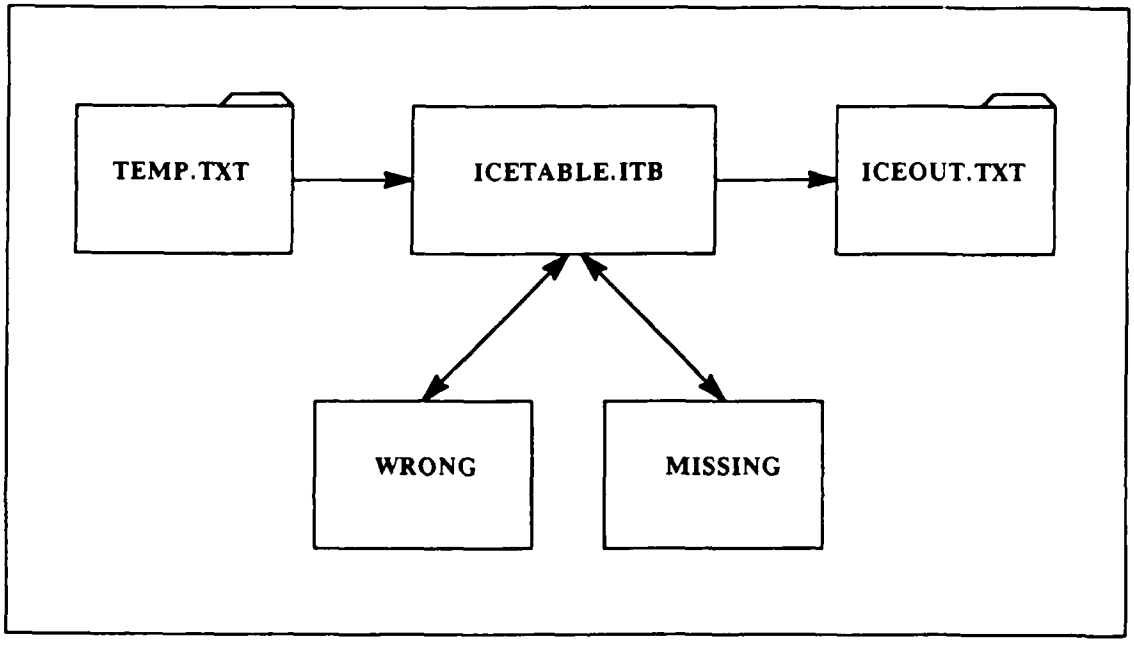


Figure 12. Modified Prototype System Layout

The last piece needed for the extended prototype was the definition of the rule set (Appendix E) needed by the expert system. The rule set was used to determine whether a circuit connection is correct or not. Since a circuit consists of two types of connections, two matrices of information were formulated. Recall, there are connections which are from an external source to a TTL, and interconnections from one TTL to another TTL.

The graphics system limits the kind and number of external connections. There are 4 kinds of ports that can be connected to a TTL, they are: input, power, ground, and clock. These are connected to 5 kinds of TTL pins: input, power, ground, clock, and output. (TTL pins such as strobe, enable, etc. are treated as input pins). Each connection is labeled as legal or illegal (L or I). Note: some connections that are marked as illegal could be used in a circuit, and the circuit would function. For instance, a clock signal connected to an input leg of a TTL is marked as illegal. This could be legitimate connection; however, the designer of the simulator identified this as an illegal connection. Other design constraints by the graphics system have been incorporated into the matrix depicted in Figure 13, the matrix of external sources to TTL connections.

The second kind of connection, TTL to TTL, was handled in the same manner as the external connections. Figure 14 depicts the legal connections between TTLs. After the rule set was defined, development of the extended prototype began.

3.4 Extended Prototype

The first order of business was cleaning up the interface between the graphics system and the interface. The elimination of TEMPTBL1.ITB from the input sequence was easily done. Guru allows files to be loaded into tables with one command; therefore, ICETABLE was loaded directly from the input file. The output files were a little more trouble. Additional tables were developed to handle the error messages. These tables were used to write to ASCII files upon completion of processing.

External Connection	TTL values: Input	Power	Ground	Clock	Output
Input	L	L	L	L	I
Power	L	L	I	I	I
Ground	L	I	L	I	I
Clock	I	I	I	L	I

Figure 13. Matrix of External Sources to TTL Connections

		TTL values				
		Input	Power	Ground	Clock	Output
TTL values	Input	I	I	I	I	L
	Power	I	I	I	I	I
	Ground	I	I	I	I	I
	Clock	I	I	I	I	I
	Output	L	I	I	I	I

Figure 14. Matrix of TTL to TTL Connections

The biggest hurdle for the extended prototype was the creation of the rule base. Using the two matrices (Figures 13 and 14), 33 rules were created (Appendix E). The rules fire in a forward-chaining fashion using a numeric priority system. Once the goal is reached execution stops and Guru continues processing the next request. A goal is reached whenever a connection is determined to be legal or illegal. Each rule has a priority associated with its importance. Guru automatically calculates the priority and executes accordingly.

The priorities range from 80 - 95, and were established in a straightforward fashion. The highest priority is given to the rules which decide the kind of connection being considered. This information determines which matrix of connections will be used to evaluate the present connection. Since the majority of connections of a circuit are TTL interconnections (the input of one TTL tied to the output of another TTL), it would be detrimental to the processing speed to make this the lowest priority; therefore, this is one of the first connections checked. On the other hand, it was not enough to verify that an input is connected to an output; what if this connection was to the same gate on the same chip? This violates a digital design principle by creating a race condition. Therefore, along with errors associated with the connectivity of the TTLs, errors such as race conditions and chip fanout were checked.

Once the ruleset was completed, the extended prototype was ready for testing. The same sequence of test circuits used for the initial prototype were used on the extended prototype. No errors were encountered, the extended prototype was fully operational in a stand-alone mode.

Figure 15 is the expert system program module flowchart. The labels without a suffix are perform files. The perform files use tables, files, and other structures. The meaning of each suffix is listed below:

- .itb = table.
- .txt = file.
- .ind = index file.
- .rss = rule set.
- .rsc = compiled rule set.

Chapter 4 of this thesis discusses the performance of the extended prototype given simple and complex circuits. The next section is the last stage, integration of the entire expert system into the engineering workstation.

3.5 Delivery System

Delivery System is the last stage of the Citrenbaum expert system development methodology. Integration into the workstation, and verification of the design specific requirements was accomplished using the Z-248 microcomputer targeted for use by the engineering students taking digital circuit design in the Fall Quarter 1987.

The expert system worked in a stand-alone mode; however, problems occurred when the integration took place. Guru would overwrite several pages in memory that the graphics system required for operation. The solution was a modification to the graphics system and the development of a driver program to orchestrate the execution of the workstation. By using a driver program, DOS was burdened with swapping the memory in/out as each system was used. The driver program would execute whichever facet of the workstation the user desired.

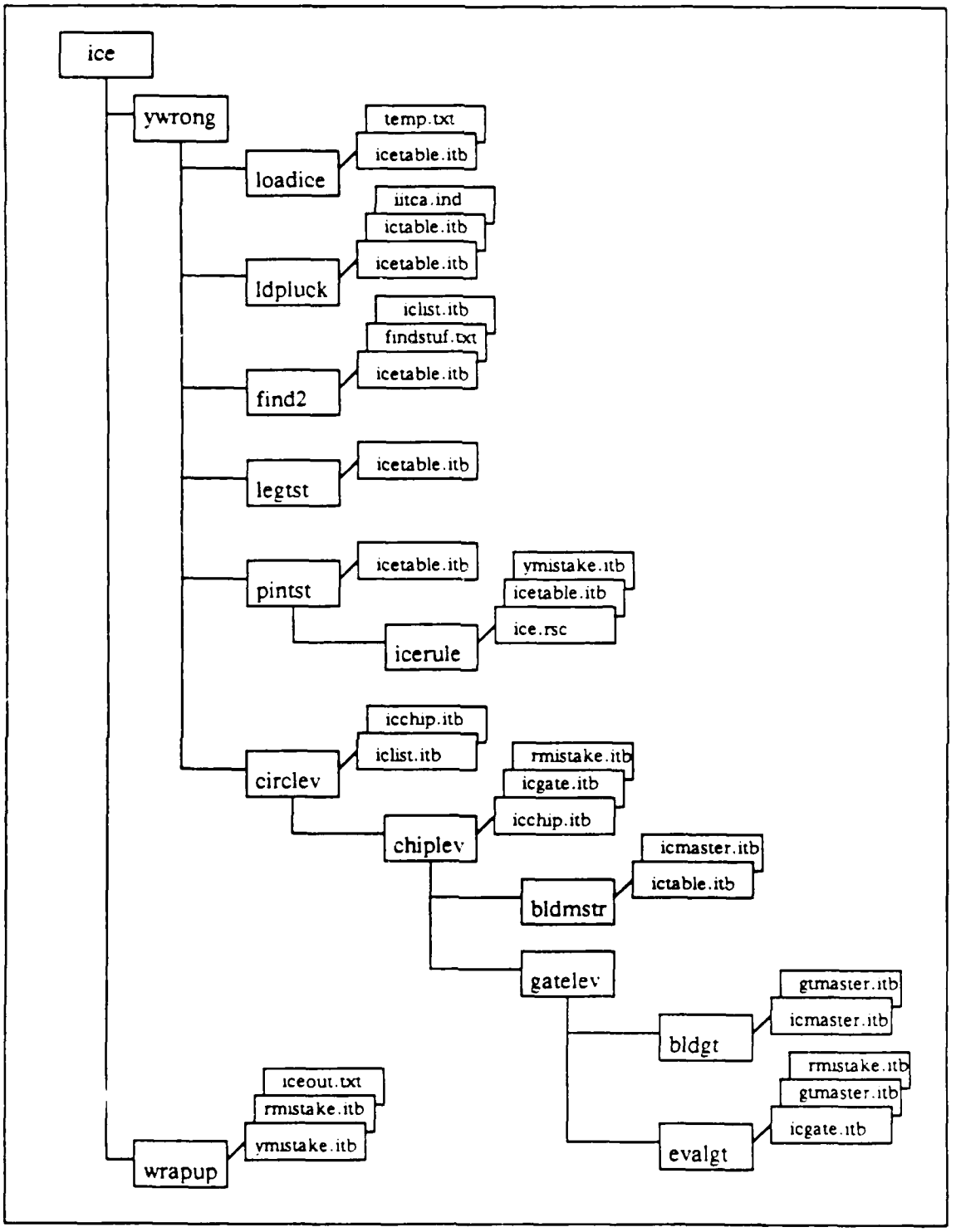


Figure 15. Expert System Program Module Flowchart

The end result was an integrated engineering workstation that allowed the user to graphically design a circuit, submit it to an expert system for analysis, then pass it to a simulator to observe the results. All three systems work in concert or in a stand-alone mode.

4. Results

4.1 Performance Characteristics

The overall performance can best be described as methodical. The solution is straightforward and uses structured programming techniques to evaluate a circuit. The expert system tool can be partitioned to execute some or all of the operations. The interconnection of the programming modules permits execution of a specific phase such as input or output. By segmenting the code into modules, I was able to make changes much easier, run and test specific attributes of the system, and also partition the system for timing specific modules. The timing yielded some interesting results.

First small circuits, circuits with 2-3 TTLs and less than 10 total gates were I/O victims within the system. The expert system exhibited a 40 second cost just for execution. A simple chip with one gate takes about 46 seconds to execute.

Using larger circuits, such as 8 TTLs with 15 or more gates, causes the system to reach a 'steady state.' The system takes approximately 75% of the total processing time just to find the missing legs. The other 25% of the time is spent doing I/O on the circuit (approximately 12% of the total time) and executing the rule set for wrong connections (approximately 13% of the time). Figure 16 depicts the actual percentages of total time spent analyzing a test circuit comprised of 8 TTLs with 16 gates utilized. In this case, 75.97% of the total time was spent finding the missing legs. The percentage of time spent on wrong legs was 11.69%, and the remaining 12.34% is attributed to the I/O of the circuit.

Appendix D contains the results of all the test cases used to evaluate the expert system. Figure 17 is a composite of the test data in Appendix D. Note that the percentage of time spent on finding the missing legs is consistently 70 - 80 percent of the total time across a wide range of TTLs and number of gates. Despite the

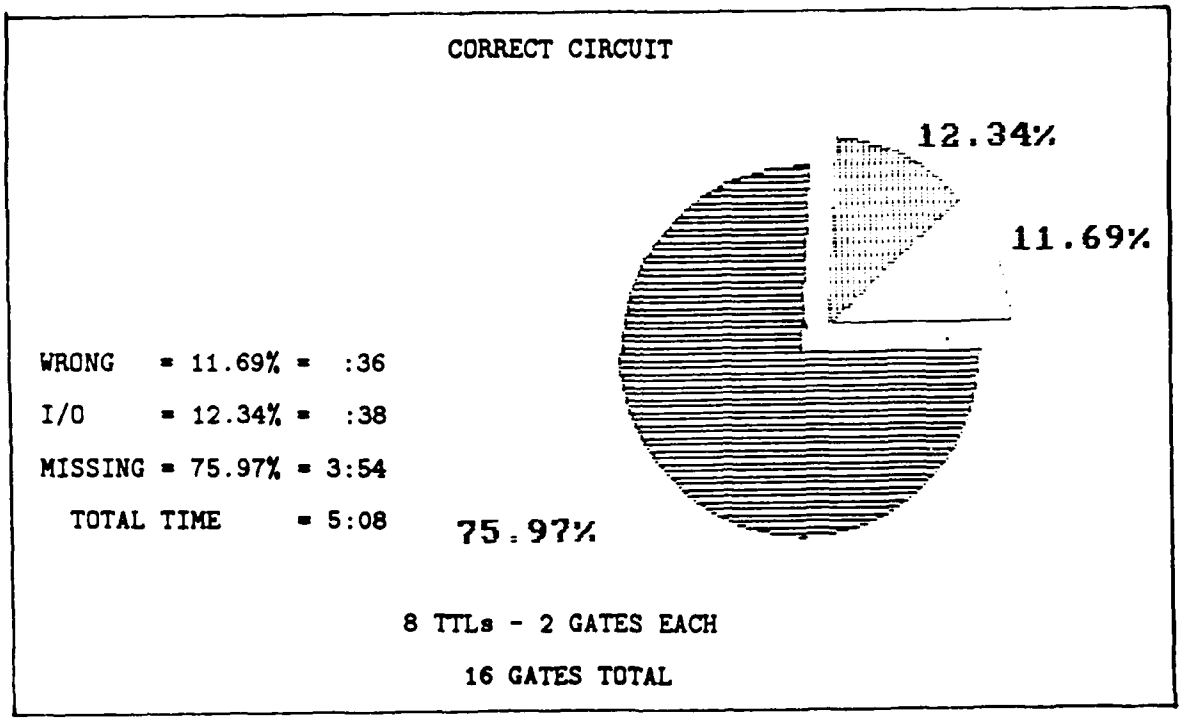


Figure 16. Workload Pie Chart of a Single Test Case

introduction of 10% errors in each test case, the time spent on finding the missing legs stays in the 70 - 80 percent range.

There are several reasons for the long delay in finding the missing legs. This is due primarily to the time spent read/writing to the disk. The missing function uses tables and database functions to find the missing connections. Guru uses the disk extensively during table manipulation. The algorithm used to find a missing leg contributes to the delay because it matches 'one to many' for every gate connection evaluated. If a gate has 3 connections, each one is checked individually.

Figure 18 shows the time spent on finding the wrong legs added to the time spent on missing legs. Once again the percentage of time spent on wrong legs is consistent throughout the suite of test cases. The remaining percentage of time (approximately 10 - 15 percent) is the time spent on circuit I/O.

The overall performance of expert system appears to function in a linear fashion, as the number of gates or number of chips is increased, there is a linear increase

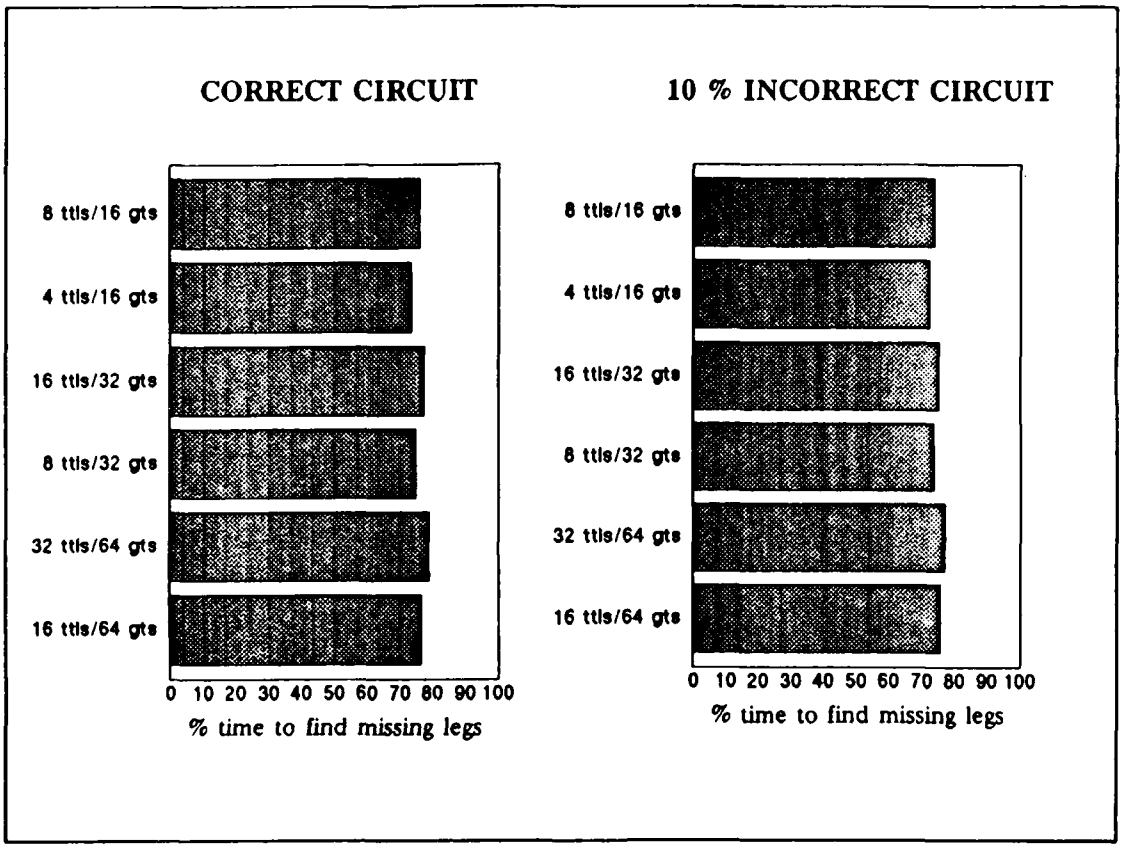


Figure 17. Workload Graph for Finding Missing Legs

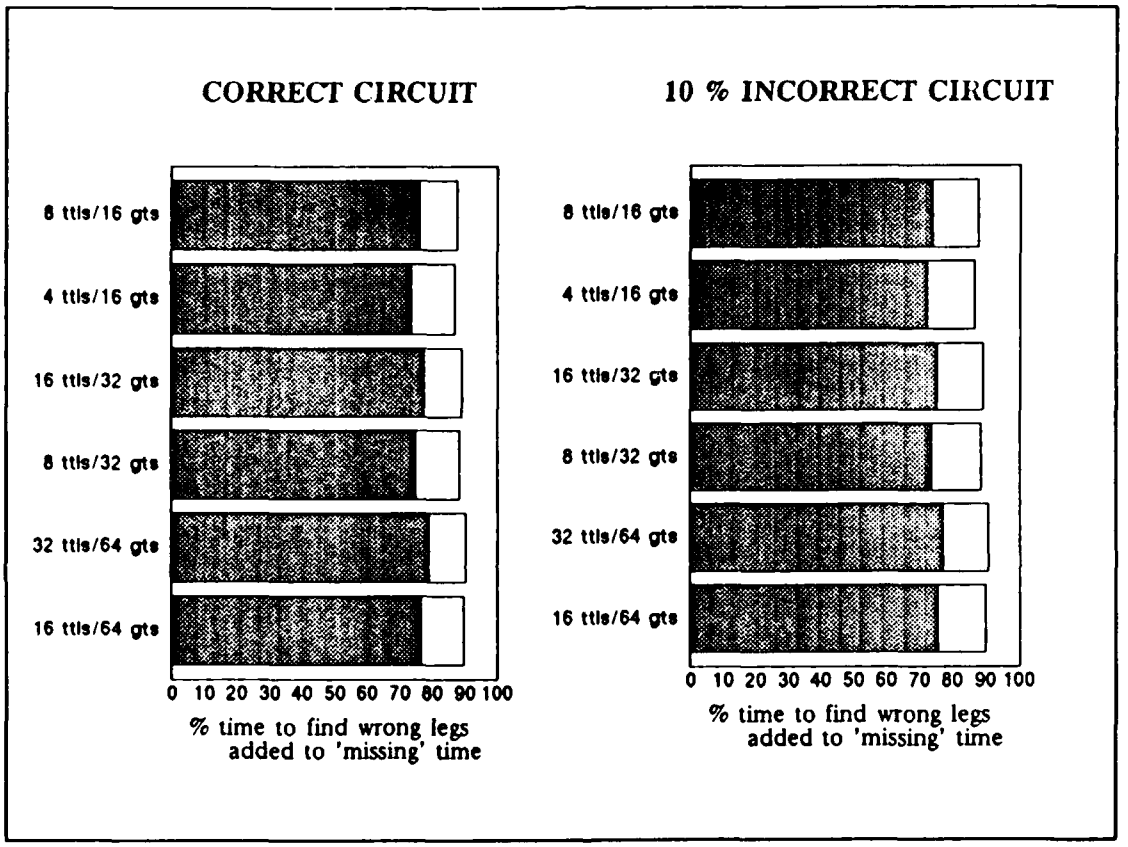


Figure 18. Workload Graph for Finding Missing and Wrong Legs

in time. For example, if a circuit is designed using 8 - SN7400s, 2 gates each, the performance time will be around 5 minutes. Doubling the number of chips to 16, the performance time will be just under 10 minutes. Doubling the count again to 32, the time would be around 20 minutes. Figure 19 illustrates the linear feature of the expert system.

Viewing the performance from another perspective, if a circuit is built using 32 chips with 2 gates each, taking approximately 20 minutes to execute, what would the effect be if 16 chips were used with 4 gates? Classical digital design advocates using all the gates available before another chip is added. In this case, the 16 chips with 4 gates each executed in approximately 17 minutes.

The point is, if good design techniques are used (such as using all the gates before another chip is added), the expert system will execute in a shorter period of time. If poor design techniques are used, longer execution times occur. These times reflect circuits with no wiring mistakes. If there were 10% wiring mistakes in the 32 chip design, then add 90 seconds (approximately 7% delay). The delay in the 16 chip design is 44 seconds (approximately 5% delay).

Additional testing examined the specific effects of using database functions in concert with expert system rules. For instance, given a circuit comprised of 6 TTLs and 75 pin connections, the time required to find all the wrong legs using only the expert system was 122 seconds. Using the same circuit, but first executing a database comparison operation, reduces the total time to 45 seconds (10 seconds for the database operation, 35 seconds for the expert system.) The database operation reduces the list of pin connections by removing all the correct connections; therefore, the expert system has a much smaller list to execute. These results were verified across the entire set of test cases.

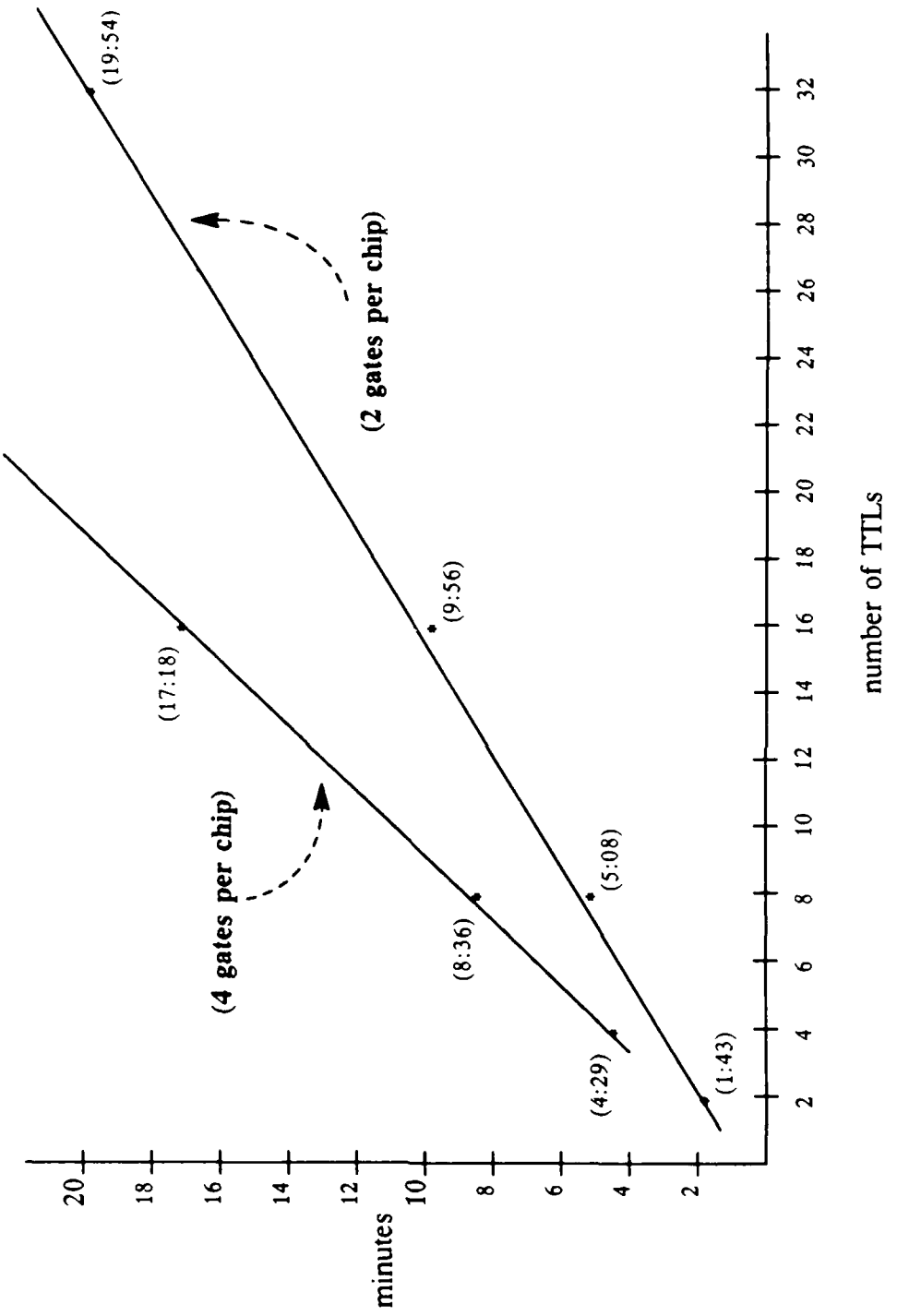


Figure 19. Performance Characteristics (time vs TTLs)

4.2 Assessment of Surveys

A survey was completed by the students using the engineering workstation to design digital circuits for EENG 450. Questions were directed at each component of the workstation, at the workstation as a whole, and some general questions regarding the background of the user.

The majority of the users were familiar with how to use a microcomputer, but none had ever used a CAD tool before. Most users were familiar with digital logic design. The responses to the expert system were favorable. Every survey reflected that the expert system provided a correct and complete analysis of the circuit, finding both wrong and missing legs. Some comments were directed at the speed of the expert system. As discussed earlier, the processing time is linked to the size of the circuit; therefore, as the circuit grows in size, the analysis will increase proportionally in time.

The expert system can be executed at any time during design to help the user verify each piece as it is built. Most of the users waited until the entire circuit was designed before invoking the expert system. Many of the users were not aware that the expert system could be used incrementally as the circuit is designed.

Comments regarding the workstation as a whole were also favorable. Many of the users would prefer to use the workstation to design digital circuits rather than design the circuits, by hand, on a circuit board. All agreed that the workstation was easy to use and was helpful.

5. *Conclusions*

5.1 Prototype Evaluation

The biggest lesson learned from this thesis effort was the importance of designing a system before it's implemented. Tackling this thesis like a big programming assignment caused numerous false starts and wasted energy. By first decomposing the problem into manageable pieces, the framework for the thesis began to take shape. Several weeks were spent designing and testing different Guru structures. The actual implementation of the prototype system took approximately three weeks.

My experience with database structures and functions has grown enormously; I now realize the power of using 'spreadsheets.' The expert system part of the thesis was easier to design because of my familiarity with the subject. Certainly a novice to AI or expert systems would have a tougher time of understanding the structures. However, Guru would be a tremendous help because of the help menu's and documentation.

By defining the user interfaces early in system development, we were able to pass files and communicate in DOS long before the individual systems were built. Having the file formats and user interfaces defined made the I/O portions easier to implement.

Overall, I was pleased with the results of the entire engineering workstation. The feedback from the surveys given to the students using the digital lab was positive. It's good to see that the system is assisting students in designing digital circuits.

5.2 Summary Conclusion

This thesis demonstrated how basic engineering techniques can be used to decompose a complex problem into smaller, solvable pieces. The performance issues, raised in the previous chapter, can be resolved by better technology. A linear increase

in the processing speed should cause a proportional reduction in the processing time since the expert system operates in a linear fashion. The use of database structures add the needed power for handling large amounts of data. The combination of the rule set and database functions provide a powerful tool for solving complex problems.

5.3 Future Research

Future research is warranted in two avenues, performance speed and application. Each avenue is not mutually exclusive. As the performance is increased, more applications may be investigated and in some applications the performance speed may not be relevant.

5.3.1 Performance Speed. The speed of the expert system could be dramatically increased by using different hardware. For instance, a ram disk would eliminate the time the disk uses to process the missing legs. Aside from a hardware change, an interface could be designed in Guru which would allow the user to identify specific chips for evaluation (instead of the entire circuit). This would eliminate redundant checking of the same chips. Also, an interface could be designed to evaluate a circuit for only wrong legs. Since missing legs takes approximately 75% of the total time to evaluate a circuit, just checking for wrong legs would save 75% of the total processing time. By directing the scope of the expert system other applications can be examined.

5.3.2 Application. The current expert system evaluates digital circuits by decomposing them into chips than the chips into gates. This could be raised one level to an entire device; where, the device is comprised of multiple circuits which would be decomposed into chips. This same algorithm could apply for decomposing VLSI circuits. The key point is, by letting the user direct the workload, this algorithm can be applied across a wide range of applications.

Appendix A. *Engineering Workstation Chip Library*

CHIP DESCRIPTION

7400 - Quad 2-Input positive NAND Gate.

7402 - Quad 2-Input positive NOR Gate.

7404 - Hex Inverters.

7408 - Quad 2-Input positive AND Gate.

7410 - Triple 3-Input positive NAND Gate.

7420 - Dual 4-Input positive NAND Gate.

7425 - Dual 4-Input positive NOR Gate with strobe.

7427 - Triple 3-Input positive NOR Gate.

7430 - Single 8-Input positive NAND Gate.

7442 - Single BCD to DECIMAL (4 line to 10 line Decoder).

7483 - Single 4-Bit Binary Full adder with fast carry.

7486 - Quad 2-Input Exclusive-OR Gate.

7489 - Single 64-Bit Read/Write Memory.

7493 - Single 4-Bit binary counter.

7495 - Single 4-Bit Shift Register.

74107 - Dual J-K Flip Flop with clear.

74109 - Dual J-K positive edge-triggered Flip Flop with preset and clear.

74116 - Dual 4-Bit Latches.

74135 - Quad Exclusive-OR/NOR Gates.

74151 - Single 1-of-8 Data Selector/multiplexer.

74153 - Dual 4-Line to 1-Line Data Selector/multiplexer.

74157 - Quad 2 to 1-Line Data Sel/mult (Non-Inverted Data Outputs).

74163 - Synchronous 4-Bit Counter (Binary, synchronous clear).

74175 - Quad D-Type Flip Flop.

74181 - Arithmetic Logic Units/Function Generators.

74183 - Dual Carry Save Full Adders.

74193 - Synchronous Up/Down Dual Clock Counter (Binary with clear).

74194 - Single 4-Bit Bidirectional Universal Shift Register.

74274 - Single 4-Bit by 4-Bit Binary Multiplier.

74279 - Quad (Inv)S - (Inv)R Latch.

74284 - Single 4-Bit by 4-Bit Parallel Binary Multiplier
used with '285'.

74285 - Single 4-Bit by 4-Bit Parallel Binary multiplier
used with '284'.

74298 - Quad 2-Input Multiplexer with storage.

74378 - Hex D-Type Flip Flop.

Appendix B. *Database Library*

CHIP	PIN	GATE	VALUE	CHIP DESCRIPTION
007400	01	a	input	
007400	02	a	input	
007400	03	a	output	
007400	04	b	input	
007400	05	b	input	
007400	06	b	output	
007400	07	chip	ground	quad 2-input nand
007400	08	c	output	
007400	09	c	input	
007400	10	c	input	
007400	11	d	output	
007400	12	d	input	
007400	13	d	input	
007400	14	chip	power	quad 2-input nand
007402	01	a	output	
007402	02	a	input	
007402	03	a	input	
007402	04	b	output	
007402	05	b	input	
007402	06	b	input	
007402	07	chip	ground	quad 2-input pos nor gate
007402	08	c	input	
007402	09	c	input	

```
007402    10    c       output
007402    11    d       input
007402    12    d       input
007402    13    d       output
007402    14    chip    power     quad 2-input pos nor gate

007404    01    a       input
007404    02    a       output
007404    03    b       input
007404    04    b       output
007404    05    c       input
007404    06    c       output
007404    07    chip    ground    hex inverters
007404    08    d       output
007404    09    d       input
007404    10    e       output
007404    11    e       input
007404    12    f       output
007404    13    f       input
007404    14    chip    power     hex inverters

007408    01    a       input
007408    02    a       input
007408    03    a       output
007408    04    b       input
007408    05    b       input
007408    06    b       output
007408    07    chip    ground    quad 2-input pos and gate
```

007408	08	c	output	
007408	09	c	input	
007408	10	c	input	
007408	11	d	output	
007408	12	d	input	
007408	13	d	input	
007408	14	chip	power	quad 2-input pos and gate
007410	01	a	input	
007410	02	a	input	
007410	03	b	input	
007410	04	b	input	
007410	05	b	input	
007410	06	b	output	
007410	07	chip	ground	triple 3-input pos nand gate
007410	08	c	output	
007410	09	c	input	
007410	10	c	input	
007410	11	c	input	
007410	12	a	output	
007410	13	a	input	
007410	14	chip	power	triple 3-input pos nand gate
007420	01	a	input	
007420	02	a	input	
007420	03	nc	nc	
007420	04	a	input	
007420	05	a	input	

007420	06	a	output	
007420	07	chip	ground	dual 4-input pos nand gate
007420	08	b	output	
007420	09	b	input	
007420	10	b	input	
007420	11	nc	nc	
007420	12	b	input	
007420	13	b	input	
007420	14	chip	power	dual 4-input pos nand gate
007425	01	a	input	
007425	02	a	input	
007425	03	a	input	
007425	04	a	input	
007425	05	a	input	
007425	06	a	output	
007425	07	chip	ground	dual 4-inpt pos nor gte w/stro
007425	08	b	output	
007425	09	b	input	
007425	10	b	input	
007425	11	b	input	
007425	12	b	input	
007425	13	b	input	
007425	14	chip	power	dual 4-inpt pos nor gte w/str
007427	01	a	input	
007427	02	a	input	
007427	03	b	input	

007427	04	b	input	
007427	05	b	input	
007427	06	b	output	
007427	07	chip	ground	triple 3-input pos nor gate
007427	08	c	output	
007427	09	c	input	
007427	10	c	input	
007427	11	c	input	
007427	12	a	output	
007427	13	a	input	
007427	14	chip	power	triple 3-input pos nor gate
007430	01	a	input	
007430	02	a	input	
007430	03	a	input	
007430	04	a	input	
007430	05	a	input	
007430	06	a	input	
007430	07	chip	ground	8-input pos nand gate
007430	08	a	output	
007430	09	nc	nc	
007430	10	nc	nc	
007430	11	a	input	
007430	12	a	input	
007430	13	nc	nc	
007430	14	chip	power	8-input pos nand gate
007442	01	a	output	

007442	02	a	output	
007442	03	a	output	
007442	04	a	output	
007442	05	a	output	
007442	06	a	output	
007442	07	a	output	
007442	08	chip	ground	4 to 10 decoder-bcd to dec
007442	09	a	output	
007442	10	a	output	
007442	11	a	output	
007442	12	a	input	
007442	13	a	input	
007442	14	a	input	
007442	15	a	input	
007442	16	chip	power	4 to 10 decoder-bcd to dec

007483	01	a	input	
007483	02	a	output	
007483	03	a	input	
007483	04	a	input	
007483	05	chip	power	4 bit bin full adder w/carry
007483	06	a	output	
007483	07	a	input	
007483	08	a	input	
007483	09	a	output	
007483	10	a	input	
007483	11	a	input	
007483	12	chip	ground	4 bit bin full adder w/carry

007483	13	a	input	
007483	14	a	output	
007483	15	a	output	
007483	16	a	input	
007486	01	a	input	
007486	02	a	input	
007486	03	a	output	
007486	04	b	input	
007486	05	b	input	
007486	06	b	output	
007486	07	chip	ground	2-input exclusive or gate
007486	08	c	output	
007486	09	c	input	
007486	10	c	input	
007486	11	d	output	
007486	12	d	input	
007486	13	d	input	
007486	14	chip	power	2-input exclusive or gate
007489	01	a	input	
007489	02	a	input	
007489	03	a	input	
007489	04	a	input	
007489	05	a	output	
007489	06	a	input	
007489	07	a	output	
007489	08	chip	ground	64 bit r/w memory

007489	09	a	output	
007489	10	a	input	
007489	11	a	output	
007489	12	a	input	
007489	13	a	input	
007489	14	a	input	
007489	15	a	input	
007489	16	chip	power	64 bit r/w memory
007493	01	chip	clock	4 bit binary counter
007493	02	a	input	
007493	03	a	input	
007493	04	nc	nc	
007493	05	chip	power	4 bit binary counter
007493	06	nc	nc	
007493	07	nc	nc	
007493	08	a	output	
007493	09	a	output	
007493	10	chip	ground	4 bit binary counter
007493	11	a	output	
007493	12	a	output	
007493	13	nc	nc	
007493	14	chip	clock	4 bit binary counter
007495	01	a	input	
007495	02	a	input	
007495	03	a	input	
007495	04	a	input	

007495	05	a	input	
007495	06	a	input	
007495	07	chip	ground	parallel i/o shift r/l ser in
007495	08	chip	clock	parallel i/o shift r/l ser in
007495	09	chip	clock	parallel i/o shift r/l ser in
007495	10	a	output	
007495	11	a	output	
007495	12	a	output	
007495	13	a	output	
007495	14	chip	power	parallel i/o shift r/l ser in
074107	01	a	input	
074107	02	a	output	
074107	03	a	output	
074107	04	a	input	
074107	05	b	output	
074107	06	b	output	
074107	07	chip	ground	dual j k ff w/clear
074107	08	b	input	
074107	09	b	clock	
074107	10	b	input	
074107	11	b	input	
074107	12	a	clock	
074107	13	a	input	
074107	14	chip	power	dual j k ff w/clear
074109	01	a	input	
074109	02	a	input	

074109	03	a	input	
074109	04	a	clock	
074109	05	a	input	
074109	06	a	output	
074109	07	a	output	
074109	08	chip	ground	j k pos edge w/preset & clr
074109	09	b	output	
074109	10	b	output	
074109	11	b	input	
074109	12	b	clock	
074109	13	b	input	
074109	14	b	input	
074109	15	b	input	
074109	16	chip	power	j k pos edge w/preset & clr

074116	01	a	input	
074116	02	a	input	
074116	03	a	input	
074116	04	a	input	
074116	05	a	output	
074116	06	a	input	
074116	07	a	output	
074116	08	a	input	
074116	09	a	output	
074116	10	a	input	
074116	11	a	output	
074116	12	chip	ground	dual 4 bit latches
074116	13	b	input	

074116	14	b	input	
074116	15	b	input	
074116	16	b	input	
074116	17	b	output	
074116	18	b	input	
074116	19	b	output	
074116	20	b	input	
074116	21	b	output	
074116	22	b	input	
074116	23	b	output	
074116	24	chip	power	dual 4 bit latches
074135	01	a	input	
074135	02	a	input	
074135	03	a	output	
074135	04	chip	input	exclusive or/nor gate
074135	05	b	input	
074135	06	b	input	
074135	07	b	output	
074135	08	chip	ground	exclusive or/nor gate
074135	09	c	output	
074135	10	c	input	
074135	11	c	input	
074135	12	chip	input	exclusive or/nor gate
074135	13	d	output	
074135	14	d	input	
074135	15	d	input	
074135	16	chip	power	exclusive or/nor gate

074151	01	a	input	
074151	02	a	input	
074151	03	a	input	
074151	04	a	input	
074151	05	a	output	
074151	06	a	output	
074151	07	chip	input	sing 1 of 8 data sel/mult
074151	08	chip	ground	sing 1 of 8 data sel/mult
074151	09	a	input	
074151	10	a	input	
074151	11	a	input	
074151	12	a	input	
074151	13	a	input	
074151	14	a	input	
074151	15	a	input	
074151	16	chip	power	sing 1 of 8 data sel/mult

074153	01	a	input	
074153	02	chip	input	dual 4 to 1 data sel/mult
074153	03	a	input	
074153	04	a	input	
074153	05	a	input	
074153	06	a	input	
074153	07	a	output	
074153	08	chip	ground	dual 4 to 1 data sel/mult
074153	09	b	output	
074153	10	b	input	

074153	11	b	input	
074153	12	b	input	
074153	13	b	input	
074153	14	chip	input	dual 4 to 1 data sel/mult
074153	15	b	input	
074153	16	chip	power	dual 4 to 1 data sel/mult
074157	01	chip	input	quad 2 to 1 data sel/mult
074157	02	a	input	
074157	03	b	input	
074157	04	chip	output	quad 2 to 1 data sel/mult
074157	05	a	input	
074157	06	b	input	
074157	07	chip	output	quad 2 to 1 data sel/mult
074157	08	chip	ground	quad 2 to 1 data sel/mult
074157	09	chip	output	quad 2 to 1 data sel/mult
074157	10	b	input	
074157	11	a	input	
074157	12	chip	output	quad 2 to 1 data sel/mult
074157	13	b	input	
074157	14	a	input	
074157	15	chip	input	quad 2 to 1 data sel/mult
074157	16	chip	power	quad 2 to 1 data sel/mult
074163	01	chip	input	sync 4 bit bin ctr-sync clr
074163	02	chip	clock	sync 4 bit bin ctr-sync clr
074163	03	a	input	
074163	04	a	input	

074163	05	a	input	
074163	06	a	input	
074163	07	chip	input	sync 4 bit bin ctr-sync clr
074163	08	chip	ground	sync 4 bit bin ctr-sync clr
074163	09	chip	input	sync 4 bit bin clr-sync clr
074163	10	chip	input	sync 4 bit bin ctr-sync clr
074163	11	a	output	
074163	12	a	output	
074163	13	a	output	
074163	14	a	output	
074163	15	a	output	
074163	16	chip	power	sync 4 bit bin ctr-sync clr
074175	01	chip	input	quad d-type flip flop
074175	02	a	output	
074175	03	a	output	
074175	04	a	input	
074175	05	b	input	
074175	06	b	output	
074175	07	b	output	
074175	08	chip	ground	quad d-type flip flop
074175	09	chip	clock	quad d-type flip flop
074175	10	c	output	
074175	11	c	output	
074175	12	c	input	
074175	13	d	input	
074175	14	d	output	
074175	15	d	output	

074175	16	chip	power	quad d-type flip flop
074181	01	a	input	
074181	02	a	input	
074181	03	a	input	
074181	04	a	input	
074181	05	a	input	
074181	06	a	input	
074181	07	a	input	
074181	08	a	input	
074181	09	a	output	
074181	10	a	output	
074181	11	a	output	
074181	12	chip	ground	alu/function generator
074181	13	a	output	
074181	14	a	output	
074181	15	a	output	
074181	16	a	output	
074181	17	a	output	
074181	18	a	input	
074181	19	a	input	
074181	20	a	input	
074181	21	a	input	
074181	22	a	input	
074181	23	a	input	
074181	24	chip	power	alu/function generator
074183	01	a	input	

074183	02	nc	nc	
074183	03	a	input	
074183	04	a	input	
074183	05	a	output	
074183	06	a	output	
074183	07	chip	ground	dual car-sav full addr
074183	08	b	output	dual car-sav full addr
074183	09	nc	nc	
074183	10	b	output	
074183	11	b	input	
074183	12	b	input	
074183	13	b	input	
074183	14	chip	power	dual car-sav full addr
074193	01	a	input	
074193	02	a	output	
074193	03	a	output	
074193	04	chip	clock	sync up/down clock ctr
074193	05	chip	clock	sync up/down clock ctr
074193	06	a	output	
074193	07	a	output	
074193	08	chip	ground	sync up/down clock ctr
074193	09	a	input	
074193	10	a	input	
074193	11	a	input	
074193	12	a	output	
074193	13	a	output	
074193	14	a	input	

074193	15	a	input	
074193	16	chip	power	sync up/down clock ctr
074194	01	a	input	
074194	02	a	input	
074194	03	a	input	
074194	04	a	input	
074194	05	a	input	
074194	06	a	input	
074194	07	a	input	
074194	08	chip	ground	4 bit bi-directional shft reg
074194	09	a	input	
074194	10	a	input	
074194	11	chip	clock	4 bit bi-directional shft reg
074194	12	a	output	
074194	13	a	output	
074194	14	a	output	
074194	15	a	output	
074194	16	chip	power	4 bit bi-directional shft reg
074274	01	a	input	
074274	02	a	input	
074274	03	a	input	
074274	04	a	input	
074274	05	a	input	
074274	06	a	output	
074274	07	a	output	
074274	08	a	output	

074274	09	a	output	
074274	10	chip	ground	4 x 4 bit bin multiplier
074274	11	a	output	
074274	12	a	output	
074274	13	a	output	
074274	14	a	output	
074274	15	a	input	
074274	16	a	input	
074274	17	a	input	
074274	18	a	input	
074274	19	a	input	
074274	20	chip	power	4 x 4 bit bin multiplier
074279	01	a	input	
074279	02	a	input	
074279	03	a	input	
074279	04	a	output	
074279	05	b	input	
074279	06	b	input	
074279	07	b	output	
074279	08	chip	ground	quad inv s-r latch
074279	09	c	output	
074279	10	c	input	
074279	11	c	input	
074279	12	c	input	
074279	13	d	output	
074279	14	d	input	
074279	15	d	input	

074279	16	chip	power	quad inv s-r latch
074284	01	a	input	
074284	02	a	input	
074284	03	a	input	
074284	04	a	input	
074284	05	a	input	
074284	06	a	input	
074284	07	a	input	
074284	08	chip	ground	4 x 4 bin multiplier w/285
074284	09	a	output	
074284	10	a	output	
074284	11	a	output	
074284	12	a	output	
074284	13	a	input	
074284	14	a	input	
074284	15	a	input	
074284	16	chip	power	4 x 4 bin multiplier w/285
074285	01	a	input	
074285	02	a	input	
074285	03	a	input	
074285	04	a	input	
074285	05	a	input	
074285	06	a	input	
074285	07	a	input	
074285	08	chip	ground	4 x 4 bin multiplier w/284
074285	09	a	output	

074285	10	a	output	
074285	11	a	output	
074285	12	a	output	
074285	13	a	input	
074285	14	a	input	
074285	15	a	input	
074285	16	chip	power	4 x 4 bin multiplier w/284
074298	01	b	input	
074298	02	b	input	
074298	03	a	input	
074298	04	a	input	
074298	05	b	input	
074298	06	b	input	
074298	07	a	input	
074298	08	chip	ground	quad 2 inpt multiplier w/strob
074298	09	a	input	
074298	10	chip	input	quad 2-inpt multiplier w/strob
074298	11	chip	clock	quad 2-inpt multiplier w/strob
074298	12	chip	output	quad 2-inpt multiplier w/strob
074298	13	chip	output	quad 2 inpt multiplier w/strob
074298	14	chip	output	quad 2 inpt multiplier w/strob
074298	15	chip	output	quad 2 inpt multiplier w/strob
074298	16	chip	power	quad 2 inpt multiplier w/strob
074378	01	chip	input	hex d-type flip flop
074378	02	a	output	
074378	03	a	input	

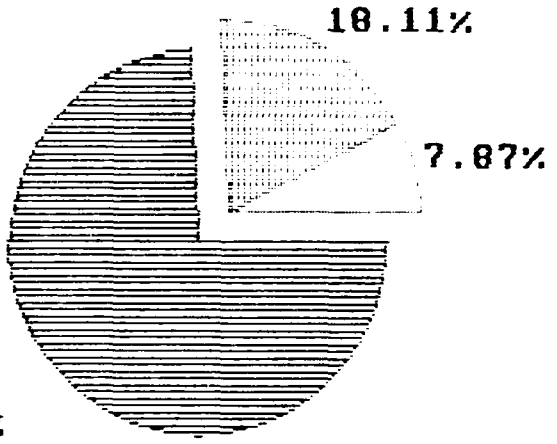
```
074378   04   b      input
074378   05   b      output
074378   06   c      input
074378   07   c      output
074378   08   chip   ground   hex d-type flip flop
074378   09   chip   clock    hex d-type flip flop
074378   10   d      output
074378   11   d      input
074378   12   e      output
074378   13   e      input
074378   14   f      input
074378   15   f      output
074378   16   chip   power    hex d-type flip flop
```

Appendix C. *Milestone Timetable*

- Mid APR - Meeting with CAD theses advisors & students.
- 15 MAY - Literature Review.
- 22 MAY - Data attribute and database requirements.
- 29 MAY - Tool selection.
- 5 JUN - Baseline testing of current prototype.
- 10 JUL - Operational system.
- 24 JUL - Verification and testing.
- 14 AUG - Evaluation.
- 25 SEP - Integration into workstation.
- 30 OCT - Final thesis draft.
- 6 NOV - Draft paper for publication.
- 13 NOV - Oral defense.
- 20 NOV - Completed thesis.
- 27 NOV - Completed paper for publication.

Appendix D. *Extended Prototype Data Sheets*

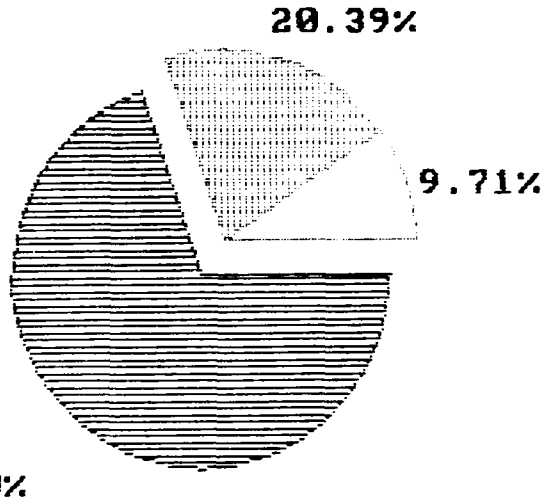
CORRECT CIRCUIT



WRONG = 7.87% = :10
I/O = 18.11% = :23
MISSING = 74.02% = 1:34
TOTAL TIME = 2:07
(time in min/sec)

4 TTLs - 1 GATE EACH

4 GATES TOTAL

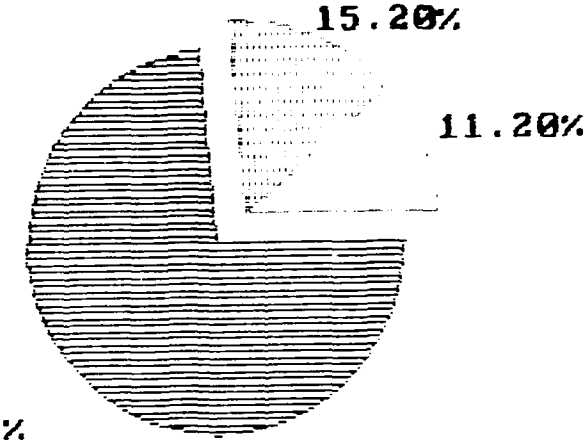


WRONG = 9.71% = :10
I/O = 20.39% = :21
MISSING = 69.90% = 1:12
TOTAL TIME = 1:43
(time in min/sec)

2 TTLs - 2 GATE EACH

4 GATES TOTAL

INCORRECT CIRCUIT



WRONG = 11.20% = :14

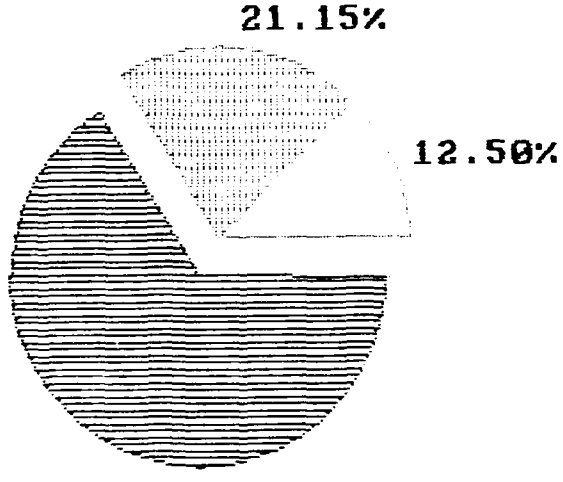
I/O = 15.20% = :19

MISSING = 73.60% = 1:32

TOTAL TIME = 2:05

4 TTLs - 1 GATE EACH

4 GATES TOTAL



WRONG = 12.50% = :13

I/O = 21.15% = :22

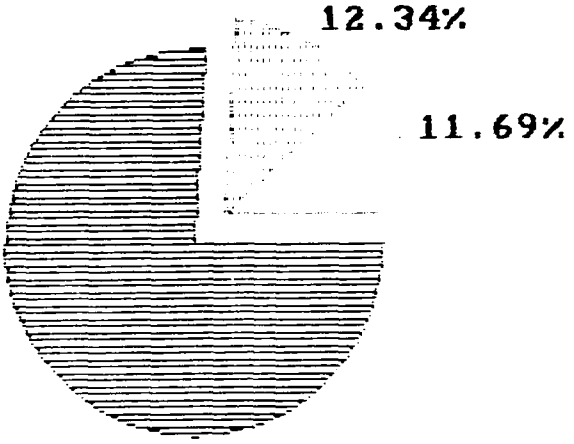
MISSING = 66.35% = 1:09

TOTAL TIME = 1:44

2 TTLs - 2 GATES EACH

4 GATES TOTAL

CORRECT CIRCUIT



WRONG = 11.69% = :36

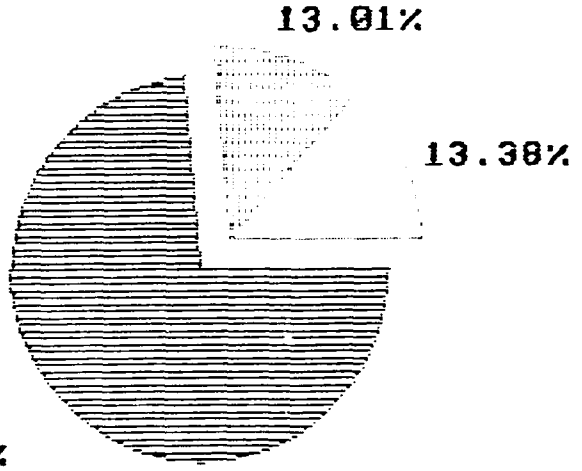
I/O = 12.34% = :38

MISSING = 75.97% = 3:54

TOTAL TIME = 5:08

8 TTLs - 2 GATES EACH

16 GATES TOTAL



WRONG = 13.38% = :36

I/O = 13.01% = :35

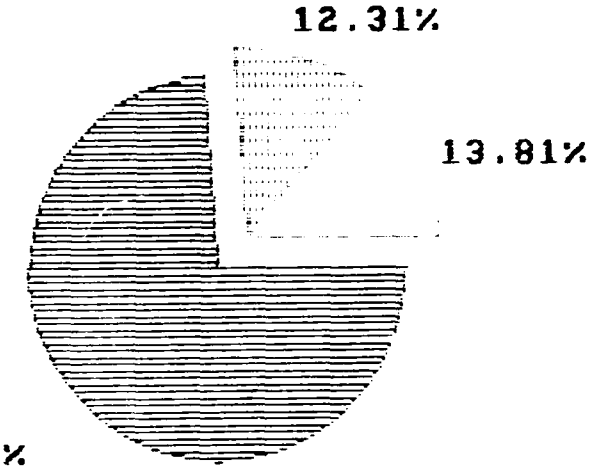
MISSING = 73.61% = 3:18

TOTAL TIME = 4:29

4 TTLs - 4 GATES EACH

16 GATES TOTAL

INCORRECT CIRCUIT



WRONG = 13.81% = :46

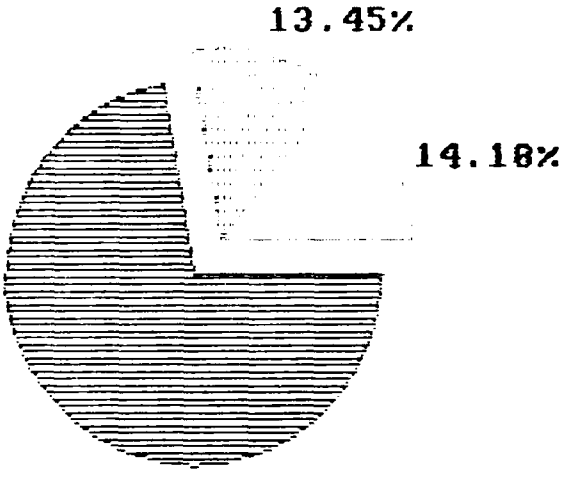
I/O = 12.31% = :41

MISSING = 73.88% = 4:06

TOTAL TIME = 5:33

8 TTLs - 2 GATES EACH

16 GATES TOTAL



WRONG = 14.18% = :39

I/O = 13.45% = :37

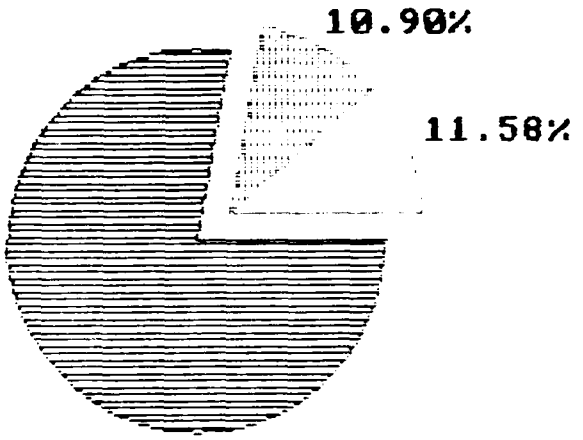
MISSING = 72.37% = 3:19

TOTAL TIME = 4:35

4 TTLs - 4 GATES EACH

16 GATES TOTAL

CORRECT CIRCUIT



WRONG = 11.58% = 1:09

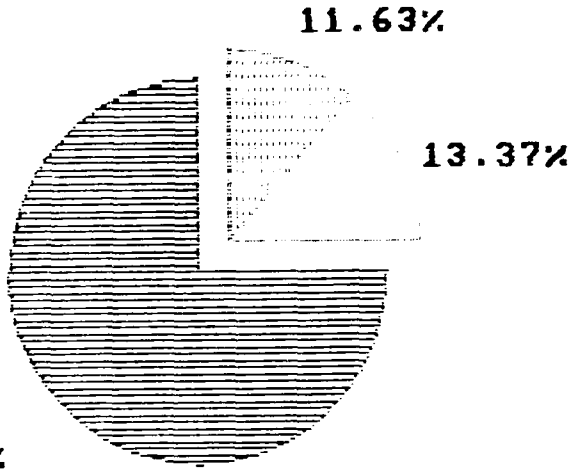
I/O = 10.90% = 1:05

MISSING = 77.52% = 7:42

TOTAL TIME = 9:56

16 TTLs - 2 GATES EACH

32 GATES TOTAL



WRONG = 13.37% = 1:09

I/O = 11.63% = 1:00

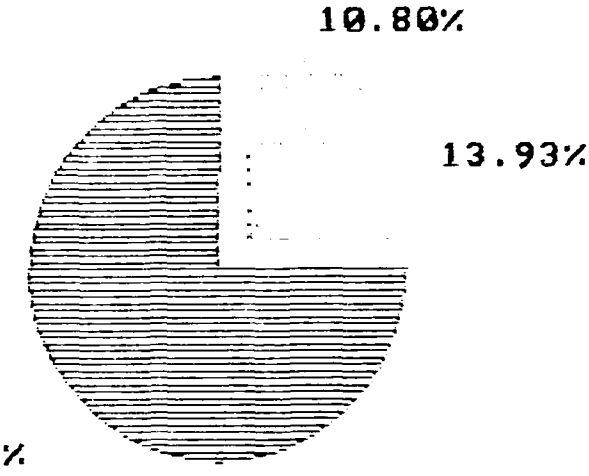
MISSING = 75.00% = 6:27

TOTAL TIME = 8:36

8 TTLs - 4 GATES EACH

32 GATES TOTAL

INCORRECT CIRCUIT



WRONG = 13.93% = 1:29

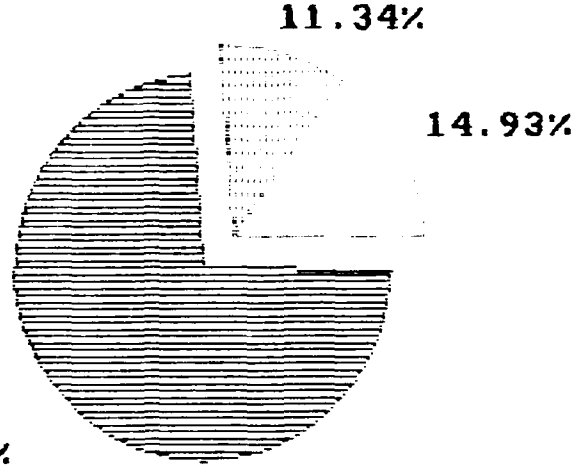
I/O = 10.80% = 1:09

MISSING = 75.27% = 8:01

TOTAL TIME = 10:39

16 TTLs - 2 GATES EACH

32 GATES TOTAL



WRONG = 14.93% = 1:19

I/O = 11.34% = 1:00

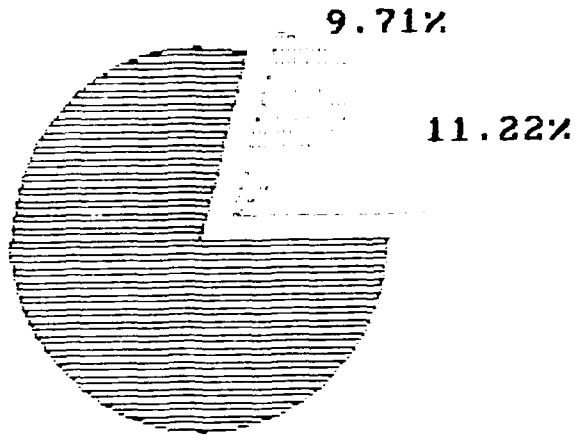
MISSING = 73.73% = 6:30

TOTAL TIME = 8:49

8 TTLs - 4 GATES EACH

32 GATES TOTAL

CORRECT CIRCUIT



WRONG = 11.22% = 2:14

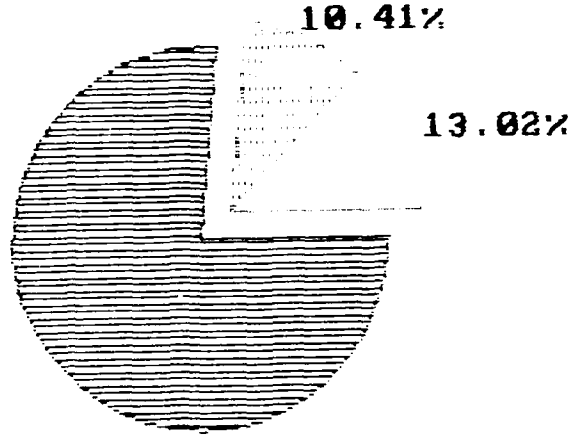
I/O = 9.71% = 1:56

MISSING = 79.07% = 15:44

TOTAL TIME = 19:54

32 TTLs - 2 GATES EACH

64 GATES TOTAL



WRONG = 13.02% = 2:16

I/O = 10.41% = 1:48

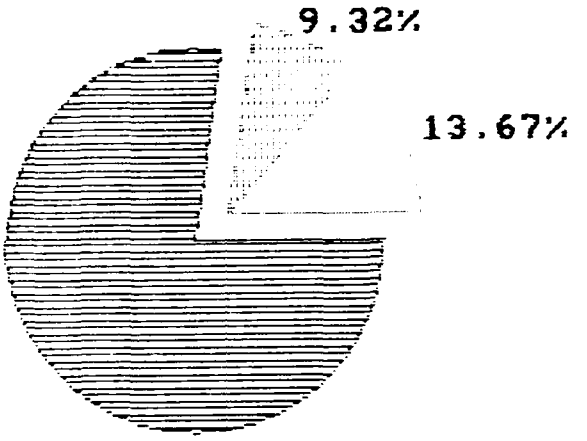
MISSING = 76.57% = 13:14

TOTAL TIME = 17:18

16 TTLs - 4 GATES EACH

64 GATES TOTAL

INCORRECT CIRCUIT



WRONG = 13.67% = 2:56

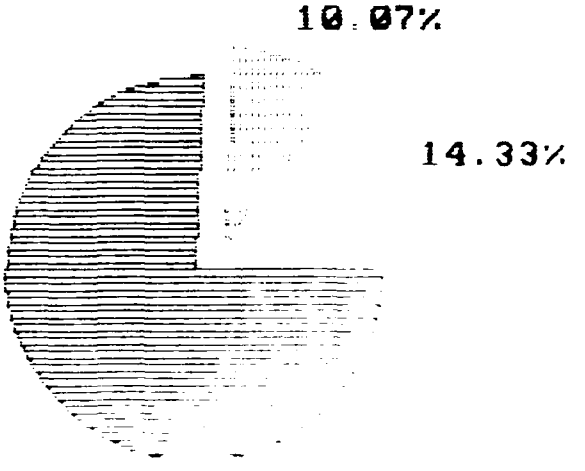
I/O = 9.32% = 2:00

MISSING = 77.01% = 16:31

TOTAL TIME = 21:27

32 TTLs - 2 GATES EACH

64 GATES TOTAL



WRONG = 14.33% = 2:35

I/O = 10.07% = 1:49

MISSING = 75.60% = 13:38

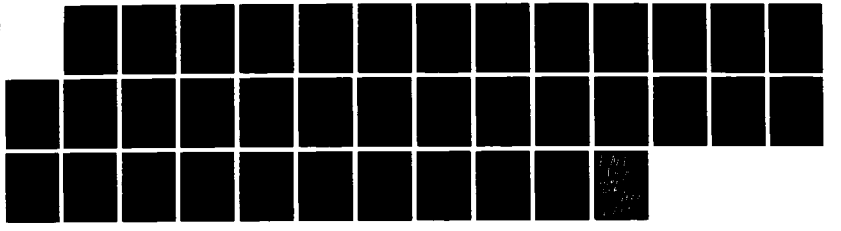
TOTAL TIME = 18:02

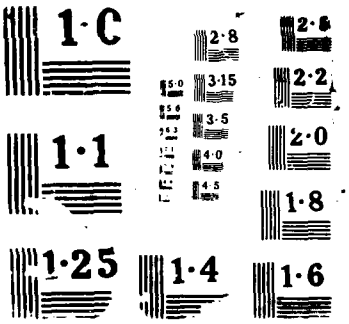
16 TTLs - 4 GATE EA

64 GATE

AD-A189 600 AN EXPERT SYSTEM FOR DISCRETE COMPONENT DIGITAL CIRCUIT DESIGN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING S M WAGNER DEC 87 2/2

UNCLASSIFIED AFIT/GCS/ENG/87D-20 F/G 9/1 NL




1·C
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0

Appendix E. *Expert System Rule Set*

```
GOAL: cktok

WINDOW:
ROW: 15
COLUMN: 5
DEPTH: 8
WIDTH: 70

INITIAL:
e.hres = 6
e.rigr = "m"
e.sord = "pu"
e.trac = "n"
e.wres = 5
e.lstr = 80
e.oprn = false
e.supd = true
cktok = unknown
fkey = substr(icetable.fpkg,1,1)
tkey = substr(icetable.tpkg,1,1)
same = false
notsame = false
samepkg = false
samechip = false
samegate = false
sameval = false
```

```
if not inuse ("ymistake") then
          use ymistake;
         endif;

DO:
finish ymistake
```

```
RULE: R1
PRIORITY: 85
IF: same and
                ((icetable.fpinval="input" and
                icetable.tpinval = "output")or
                (icetable.fpinval="output"and
                icetable.tpinval = "input"))
                and not(samechip and samegate)
THEN: change icetable.clor in icetable to
                  "k" current
        cktok = true
NEEDS: same
                icetable.fpinval
                icetable.tpinval
                samechip
                samegate
CHANGES:cktok
```

```
RULE: R2
PRIORITY: 95
IF: fkey = tkey
THEN: same = true
NEEDS: fkey
                tkey
CHANGES: same
RULE: R3
PRIORITY: 80
IF: icetable.fgate = "nc" or
                icetable.tgate = "nc"
THEN: change icetable.clor in icetable to
                  "y" current
cktok = true
NEEDS: icetable.fgate
icetable.tgate
CHANGES: cktok
RULE:   R4
        /*  removed  */
```

```
RULE: R5
PRIORITY: 95
IF: fkey <> tkey
THEN: notsame = true
NEEDS: fkey
                tkey
CHANGES: notsame
RULE: R6
PRIORITY: 90
IF: same and
                icetable.fpkg = icetable.tpkg
THEN: samechip = true
NEEDS: same
icetable.fpkg
icetable.tpkg
CHANGES: samechip
```

```
RULE: R7
PRIORITY: 90
IF: same and
                icetable.fgate = icetable.tgate
THEN: samegate = true
NEEDS: same
icetable.fgate
icetable.tgate
CHANGES: samegate
RULE: R8
PRIORITY: 90
IF: same and
                icetable.fpinval = icetable.tpinval
THEN: sameval = true
NEEDS: same
icetable.fpinval
icetable.tpinval
CHANGES: sameval
```

```
RULE: R9
PRIORITY: 90
IF: same and samechip and samegate
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r9val ="y package "+\
                icetable.fpkg+" has a race"+\
                " condition present between pins "+\
                icetable.fpin+\
                " and "+icetable.tpin
                change errfield in ymistake to
                  r9val current
cktok = true
NEEDS: same
samechip
samegate
CHANGES: cktok
```

```
RULE: R10
PRIORITY: 80
IF: same and sameval
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r10val ="y two "+\
                trim(icetable.fpinval)+"'s are tied"+\
                " together--"+"pin "+icetable.fpin+" of "+\
                icetable.fpkg+" and pin "+icetable.tpin+\
                " of "+icetable.tpkg
                change errfield in ymistake to r10val current
cktok = true
NEEDS: same
sameval
CHANGES: cktok
```

```
RULE: R11
PRIORITY: 80
IF: same and icetable.fpinval = "power" and
        icetable.tpinval in ["inp*","gro*","clo*","out*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r11val ="y pin "+icetable.fpin+", "+\
                trim(icetable.fpinval)+\
                " of "+\
                icetable.fpkg+" is incorrectly tied to "+\
                icetable.tpkg+", pin "+icetable.tpin+" a "+\
                trim(icetable.tpinval)
                change errfield in ymistake to r11val current
cktok = true
NEEDS: same
icetable.fpinval
                icetable.tpinval

CHANGES: cktok
```

```
RULE: R12
PRIORITY: 80
IF: same and icetable.fpinval = "ground" and
        icetable.tpinval in ["inp*","pow*","clo*","out*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r12val="y pin "+icetable.fpin+", "+\
                trim(icetable.fpinval)+\
                " of "+\
                icetable.fpkg+" is incorrectly tied to "+\
                icetable.tpkg+", pin "+icetable.tpin+" a "+\
                trim(icetable.tpinval)
                change errfield in ymistake to r12val current
cktok = true
NEEDS: same
icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R13
PRIORITY: 80
IF: same and icetable.fpinval = "clock" and
        icetable.tpinval in ["inp*","pow*","gro*","out*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r13val = "y pin "+\
                icetable.fpin+", "+trim(icetable.fpinval)+\
                " of "+\
                icetable.fpkg+" is incorrectly tied to "+\
                icetable.tpkg+", pin "+icetable.tpin+" a "+\
                trim(icetable.tpinval)
                change errfield in ymistake to r13val current
cktok = true
NEEDS: same
icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R14
PRIORITY: 80
IF: same and icetable.fpinval = "input" and
        icetable.tpinval in ["pow*","gro*","clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r14val = "y pin "+\
                icetable.fpin+", "+trim(icetable.fpinval)+\
                " of "+\
                icetable.fpkg+" is incorrectly tied to "+\
                icetable.tpkg+", pin "+icetable.tpin+" a "+\
                trim(icetable.tpinval)
                change errfield in ymistake to r14val current
cktok = true
NEEDS: same
icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R15
PRIORITY: 80
IF: same and icetable.fpinval = "output" and
        icetable.tpinval in ["pow*","gro*","clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r15val = "y pin "+\
                icetable.fpin+", "+trim(icetable.fpinval)+\
                " of "+\
                icetable.fpkg+" is incorrectly tied to "+\
                icetable.tpkg+", pin "+icetable.tpin+" a "+\
                trim(icetable.tpinval)
                change errfield in ymistake to r15val current
cktok = true
NEEDS: same
icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R16
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "output" and
        icetable.tpinval in ["inp*","pow*","gro*","clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r16val ="y external "+\
                trim(icetable.tpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.fpinval)+" pin of "+\
                icetable.fpkg+", pin "+icetable.fpin
                change errfield in ymistake to r16val current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R17
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["inp*","pow*","gro*","clo*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "output"
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r17val ="y external "+\
                trim(icetable.fpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.tpinval)+" pin of "+\
                icetable.tpkg+", pin "+icetable.tpin
                change errfield in ymistake to r17val current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R18
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "clock" and
        icetable.tpinval in ["inp*","clo*"]
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R19
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "clock" and
        icetable.tpinval in ["pow*","gro*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r19val ="y external "+\
                trim(icetable.tpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.fpinval)+" pin of "+\
                icetable.fpkg+", pin "+icetable.fpin
                change errfield in ymistake to r19val current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R20
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["inp*","clo*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "clock"
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R21
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["pow*","gro*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "clock"
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r21val ="y external "+\
                trim(icetable.fpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.tpinval)+" pin of "+\
                icetable.tpkg+", pin "+icetable.tpin
                change errfield in ymistake to r21val current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R22
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "ground" and
        icetable.tpinval in ["inp*","gro*"]
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R23
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "ground" and
        icetable.tpinval in ["pow*","clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r23val ="y external "+\
                trim(icetable.tpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.fpinval)+" pin of "+\
                icetable.fpkg+", pin "+icetable.fpin
                change errfield in ymistake to r23val current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R24
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["inp*","gro*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "ground"
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R25
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["pow*","clo*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "ground"
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r25val ="y external "+\
                trim(icetable.fpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.tpinval)+" pin of "+\
                icetable.tpkg+", pin "+icetable.tpin
                change errfield in ymistake to r25val current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R26
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "power" and
        icetable.tpinval in ["inp*","pow*"]
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R27
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "power" and
        icetable.tpinval in ["gro*","clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r27val ="y external "+\
                trim(icetable.tpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.fpinval)+" pin of "+\
                icetable.fpkg+", pin "+icetable.fpin
                change errfield in ymistake to r27val current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R28
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["inp*","pow*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "power"
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R29
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["gro*","clo*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "power"
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r29val ="y external "+\
                trim(icetable.fpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.tpinval)+" pin of "+\
                icetable.tpkg+", pin "+icetable.tpin
                change errfield in ymistake to r29val current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R30
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "input" and
        icetable.tpinval in ["inp*","pow*","gro*"]
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R31
PRIORITY: 80
IF: notsame and
        icetable.fpkg in ["T*"] and
        icetable.fpinval = "input" and
        icetable.tpinval in ["clo*"]
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r31val ="y external "+\
                trim(icetable.tpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.fpinval)+" pin of "+\
                icetable.fpkg+", pin "+icetable.fpin
                change errfield in ymistake to r31val current
cktok = true
NEEDS: notsame
icetable.fpkg
                icetable.fpinval
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R32
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["inp*","pow*","gro*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "input"
THEN: change icetable.clor in icetable to
                  "k" current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
```

```
RULE: R33
PRIORITY: 80
IF: notsame and
        icetable.fpinval in ["clo*"] and
        icetable.tpkg in ["T*"] and
        icetable.tpinval = "input"
THEN: change icetable.clor in icetable to
                  "y" current
                attach 1 to ymistake
                r33val ="y external "+\
                trim(icetable.fpinval)+\
                " is mistakenly "+\
                "connected to a "+\
                trim(icetable.tpinval)+" pin of "+\
                icetable.tpkg+", pin "+icetable.tpin
                change errfield in ymistake to r33val current
cktok = true
NEEDS: notsame
icetable.fpinval
                icetable.tpkg
                icetable.tpinval
CHANGES: cktok
END:
```

Bibliography

1. Adams, Charles A. Jr. *A Digital Circuit Design Environment*, MS Thesis AFIT/ENG/GCS/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, December 1987.
2. Birmingham, William P. and Daniel P. Siewiorek. "MICON: A Knowledge Based Single Board Computer Design," *21st Design Automation Conference, IEEE Transaction*, 565-571, 1984.
3. Brownston, Lee and others. *Programming Expert Systems in OPS5*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1985.
4. Citrenbaum, Ronald and others. "Selecting a Shell," *AI Expert*, Vol 2, No 9, 32-39, September 1987.
5. DeLoria, Wayne C. *A Digital Logic Simulator with Concurrent Programming Considerations*, MS Thesis AFIT/ENG/GCS/87D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, December 1987.
6. Estes, A. and others. *ICE - An InterConnect Expert*, unpublished manuscript, Department of ENG, Air Force Institute of Technology, 1986.
7. Greenfield, Joseph D. *Practical Digital Design Using IC's*. New York: John Wiley & Sons, 1977.
8. Harmon, Paul and David King. *Expert Systems: Artificial Intelligence in Business*. New York: John Wiley & Sons Inc., 1985.
9. *Hewlett Packard Electronic Measurement * Design * Computation*, Product Manual, 143-144, 1987.
10. *HIWIRE*, Product Report, WINTEK Corporation, 1-4, 1987.
11. Kelly, Van E. "The CRITTER System-Automated Critiquing of Digital Circuit Designs," *21st Design Automation Conference, IEEE Transaction*, 419-425, 1984.
12. Kernighan, Brian W., Dennis M. Richie. *The C Programming Language*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1978.
13. Mano, M. Morris. *Digital Logic and Computer Design*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.
14. McNally, C. Wayne. "Computer-Aided Circuit Analysis in the AFLC Software Support Center," *A Cadre Research Report*. Air University, Maxwell Air Force Base, Alabama, 1987.
15. Metzger, P. W. *Managing a Programming Project*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1973.

16. Micro Data Base Systems, Inc. *Guru Menu User's Guide*. Lafayette, Indiana: Micro Data Base Systems, Inc., 1985.

17. Micro Data Base Systems, Inc. *Guru Reference Manual, Volume 1*. Lafayette, Indiana: Micro Data Base Systems, Inc., 1985.

18. Moskowitz, Leonard. "Knowledge-Based Circuit Design," *AUTOTESTCON, IEEE Transaction*, 69-74, 1985.

19. OPS/83: *The Next Generation in Expert Systems Programming*, Software Manual, Production Systems Technology, 1986

20. Rich, Elaine. *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1983.

21. Sacher, Eric. "PC-Based Logic Simulator and Test Program Development Workstation," *AUTOTESTCON, IEEE Transaction*, 107-111, 1985.

22. Shaw, W. H. and George, B. L.. "Computer Aided Design in Graduate Engineering Education," *Proceedings of the 1987 Engineering & Industrial Conference*, Chicago, May, 1987.

23. *The TTL Data Book for Design Engineers*, Second Edition. Texas Instruments, 1981.

24. Waterman, Donald A. *A Guide to Expert Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1986.

Vita

First Lieutenant Steven M. Wagner was born on September 22, 1955 in Dayton, Ohio. He enlisted in the Air Force in May, 1978 and was assigned to the Air Force Data Services Center, Pentagon, Washington D. C. He was selected for the Airman Education Commissioning Program in 1981 and attended Wright State University. He graduated in 1983 with a Bachelor of Science degree in Computer Science. He attended Officer Training School and graduated in March 1984 as a Distinguished Graduate. Upon graduation, he was assigned to the 1500th Computer Services Squadron, Headquarters, Military Airlift Command. His primary duties were as a Programming Team Chief responsible for the strategic scheduling databases for all cargo and passenger aircraft. In May 1986, Lieutenant Wagner entered the Computer Systems program at the School of Engineering, Air Force Institute of Technology.

Permanent address: 2215 Russet Avenue
Dayton, Ohio 45420

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/87D-28	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION School of Engineering 6b OFFICE SYMBOL (If applicable) AFIT/ENG	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright Patterson Air Force Base OH 45433-6583	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
See Box 19

12 PERSONAL AUTHOR(S)
Steven M. Wagner, B.S., 1st Lt, USAF

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	121

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Digital Design, Expert System, Computer Aided Engineering, AI, Engineering Workstation.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

TITLE: An Expert System for Discrete Component Digital Circuit Design

Thesis Chairman: Wade H. Shaw, Jr., USA
Asst Professor of Electrical Engineering

21 Dec 87
Dean for Research and Professional Development
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete*

Cont from Block 19:

The design of digital circuits is cumbersome and prone to errors. Current estimates show that a circuit designed with 5 integrated circuits has less than a 50% chance of working, due to wiring errors, when power is first applied. As the complexity increases, the probability of the circuit working on the first try decreases rapidly.

Design errors fall into two categories: wiring or logic errors. Wiring errors consist of improperly connected gates, missing connections, and violations of fanout, or race conditions. Logic errors, which are more complex to detect, require knowledge of intended circuit function.

This thesis designs, develops, and tests an expert system which decomposes digital circuits into subproblems described by database technology in order to detect wiring errors. Information needed to connect chips together is viewed as knowledge base information for the expert system. Information such as number of pins, value of each pin (input, output), type of chip, etc., are represented in a database. The interconnections between integrated circuits are evaluated within the expert system. This thesis merges an expert system tool with database techniques in a two-step approach that provides wiring assistance to a design engineer.

It is concluded that a two step process, combining the speed of database systems and power of expert systems, result in a synergistic approach to debugging digital circuits. It is demonstrated how basic engineering techniques can be used to decompose a complex problem into smaller, solvable pieces. Performance issues such as execution speed and workload distribution are investigated. The system is integrated into an engineering workstation currently used in the Department of Computer Science and Electrical Engineering at the Air Force Institute of Technology. Suggestions for future research are considered.

END

DATE

FILMED

APRIL

1988

DTIC